A SURFACE DIPOLE-BASED FRAMEWORK FOR FAST AND EFFICIENT SOLUTION OF ELECTROMAGNETIC SCATTERING PROBLEMS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

By

Doğaç Dilek

July 2025

A Surface Dipole-Based Framework for Fast and Efficient Solution of Electromagnetic Scattering Problems
By Doğaç Dilek
July 2025

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Vakur Behçet Ertürk(Advisor)

Mert Kalfa(Co-Advisor)

Özlem Aydın Çivi

Lale Alatan

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

A SURFACE DIPOLE-BASED FRAMEWORK FOR FAST AND EFFICIENT SOLUTION OF ELECTROMAGNETIC SCATTERING PROBLEMS

Doğaç Dilek
M.S. in Electrical and Electronics Engineering
Advisor: Vakur Behçet Ertürk
Co-Advisor: Mert Kalfa
July 2025

This thesis introduces a surface-based fast method for solving large-scale electromagnetic scattering problems within the Method of Moments (MoM) framework. Addressing the parallelization and low-frequency challenges of the Multilevel Fast Multipole Algorithm, the proposed approach represents conventional Rao-Wilton-Glisson (RWG) basis functions with uniformly distributed Hertzian dipoles placed on subdomain surfaces. This structural simplification facilitates translation operations, improves implementation efficiency, and supports parallel computing. The method comprises three stages: mapping the original current that is modeled with RWG basis functions to dipoles, translating fields via Green's function, and reconstructing local fields through inverse mapping. Numerical results are provided at both subdomain and full problem levels. Accuracy is first tested on isolated box interactions and then validated in a complete scattering problem involving a perfectly conducting sphere across different frequencies. The proposed method is benchmarked against MoM, a volumetric dipole-based approach, and the analytical Mie series solution, showing excellent agreement in all cases. Aimed to implement the method in Python with just-in-time (JIT) parallelization via Numba, the method is expected to demonstrate practical efficiency. Future work may explore symmetry-based translation reuse, machine learning for operator modeling, and GPU acceleration, making the method a promising candidate for hardware-friendly electromagnetic simulations that is strongly scalable for parallelization.

Keywords: Parallelization, Surface integral equations.

ÖZET

ELEKTROMANYETİK SAÇILMA PROBLEMLERİNE YÖNELİK HIZLI VE VERİMLİ BİR YÜZEY DİPOL YAKLAŞIMI

Doğaç Dilek
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Vakur Behçet Ertürk
İkinci Tez Danışmanı: Mert Kalfa
Temmuz 2025

Bu tez, Momentler Metodu (MoM) çerçevesinde formüle edilen büyük ölçekli elektromanyetik saçılma problemlerinin çözümüne yönelik yüzey tabanlı hızlı bir yöntem önermektedir. Çok Seviyeli Hızlı Çokkutup Yöntemi'nin paralelleştirme ve düşük frekanslardaki sınırlamalarını hedef alan bu yaklaşımda, geleneksel Rao-Wilton-Glisson (RWG) taban fonksiyonları, alt bölge yüzeylerine düzgün biçimde yerleştirilmiş Hertz dipolleri ile temsil edilir. Bu yapısal sadeleştirme, çeviri işlemlerini kolaylaştırmakta, uygulama verimliliğini artırmakta ve paralel hesaplamayı desteklemektedir. Yöntem üç aşamadan oluşur: RWG taban fonksiyonlarıyla modellenmiş özgün akımın dipollere eşlenmesi, alanların Green fonksiyonu aracılığıyla iletilmesi ve yerel alanların ters eşleme yoluyla yeniden inşası. Sayısal sonuçlar hem alt bölge hem de tüm problem düzeyinde sunulmaktadır. Doğruluk önce izole kutu etkileşimleri üzerinde test edilmiş, ardından farklı frekanslarda mükemmel iletken bir küre üzerindeki tam saçılma problemi ile doğrulanmıştır. Önerilen yöntem; MoM, hacim tabanlı bir dipol yaklaşımı ve analitik Mie serisi çözümü ile karşılaştırılmış ve tüm durumlarda yüksek düzeyde uyum göstermiştir. Python dilinde, Numba aracılığıyla Tam Zamanında (JIT) derleme destekli paralelleştirme ile geliştirilen uygulama, pratik açıdan verimli sonuçlar sunmaktadır. Gelecek çalışmalar, simetri tabanlı çeviri tekrar kullanımını, operatör modellemede makine öğrenmesi tekniklerini ve GPU hızlandırmasını araştırabilir. Bu yönüyle yöntem, donanım dostu elektromanyetik benzetimler için güçlü paralelleşebilirliğe sahip umut vadeden bir adaydır.

Anahtar sözcükler: Paralelleştirme, Yüzey integrali denklemleri.

Acknowledgement

I would like to express my deepest gratitude to my advisor, Prof. Vakur Behçet Ertürk, for his invaluable guidance, constant support, and insightful feedback throughout my research. His deep expertise and high academic standards have greatly shaped both the direction and quality of this thesis. I truly appreciate his patience, encouragement, and the freedom he allowed me to explore my ideas while always offering clear direction when needed.

I am also sincerely thankful to my co-advisor, Dr. Mert Kalfa, for his technical insight, thoughtful suggestions, and dedicated support at every stage of this work. His mentorship has been essential in overcoming challenges and refining the ideas presented in this thesis.

I would like to extend my appreciation to Prof. Özlem Aydın Çivi, my first reader, and Assoc. Prof. Lale Alatan, my second reader, for their valuable comments and helpful insights that improved the quality of this thesis.

I would also like to warmly thank Prof. Ayhan Altıntaş for kindly agreeing to serve as the substitute jury member. I deeply appreciate his support and his always encouraging presence throughout my time at the department.

Finally, I would like to thank all faculty members, colleagues, and friends who supported me throughout this academic journey. In particular, I would like to thank Enes Koç, with whom I had the pleasure of working closely under the same advisor. His collaboration, helpful discussions, and shared motivation made this journey both more productive and enjoyable.

Contents

1 Introduction **1**

2 Preliminaries **10**

2.1 Electromagnetic related preliminaries 10

2.1.1 Basics of Electromagnetic Theory 11

2.1.2 Green's Function 12

2.1.3 Surface Equivalance Principle 13

2.1.4 Electric Field Integral Equations (EFIE) 15

2.1.5 Discretization via the MoM 16

2.1.6 Rao-Wilton-Glisson (RWG) Basis Functions 18

2.1.7 Multilevel Fast Multipole Algorithm (MLFMA) 19

2.2 Coding-Related Preliminaries 24

2.2.1 Just-In-Time (JIT) Compilation 25

2.2.2 CPU Parallelization 25

2.2.3 GPU Parallelization 26

3 Performance Optimization of MLFMA 27

3.1 Just-In-Time (JIT) Compilation 27

3.2 Numba Parallelization (CPU) 30

3.3 Time Comparisons 31

3.3.1 JIT Time Results 32

3.3.2 CPU Parallelization Time Results 38

3.4 Future Directions 41

4 Surface Dipole Method 44

4.1 Formulations 44

4.1.1 Mapping to Uniform Basis Functions 46

4.1.2 Translation 54

4.1.3 Inverse Mapping to Subdomain Functions 57

4.2 Numeric Results 60

4.2.1 Volumetric and Surface Methods Comparisons for Mapping to Uniform Basis Functions 60

4.2.2 Translation and Inverse Mapping to Subdomain Functions 66

4.2.3 Numerical Results for Full-Scale Scattering Problems . . . 68

4.3 Future Work for Surface Method 76

4.3.1 Exploiting Symmetries in Translation Matrices 76

4.3.2 Machine Learning for Fast Translation Modeling 77

5 Conclusion and Future Research Directions 78

A Full Error Tables for Numerical Results of Mapping to Uniform Basis Functions 87

List of Figures

2.1 Illustration of the surface equivalence theorem applied to a perfectly conducting object. 14

2.2 Illustration of an RWG basis function. 18

2.3 Illustration of the tree structure of MLFMA. 20

2.4 Two-dimensional illustration of the spatial classification of an 8×8 computational grid in MLFMA: the test box (green), near-field region (blue), far-field region (red), and very far-field region (yellow) are identified based on hierarchical box proximity. 21

2.5 Vector decomposition between two RWGs located in the i^{th} and j^{th} far-field boxes. The vector $\bar{w}$ denotes the translation vector connecting the centers of these boxes. 22

3.1 Workflow of JIT compilation and its integration into the runtime execution pipeline. 29

3.2 Mesh representation of the test sphere using uniform triangulation. The geometry is discretized with RWG basis functions having an average edge length of 25 mm ($\lambda/48$). 33

3.3 Mesh representation of the Flamme geometry 34

3.4 Performance comparison between actual execution time and expected parallel runtime based of number cores. 40

4.1 Illustration of the object: (a) original geometry, (b) RWG discretization, and (c) Spatial decomposition of the geometry into an 8×8 grid. The central green cell denotes the selected source box. Yellow cells represent its near-field (NF) neighbors, while red cells correspond to the far-field (FF) region. Unshaded cells indicate empty boxes with no interaction. 45

4.2 (a) A group of RWG basis functions randomly located within a box, (b) uniformly distributed surface dipoles used as an alternative basis. 47

4.3 Red arrows represent Hertzian dipoles uniformly distributed on the visible surface of the cube, while green arrows denote the electric field samples computed on a surrounding sphere. The setup illustrates the far-field evaluation due to surface-based dipole sources. 50

4.4 Vector definitions used in the translation stage between two boxes in the far-field regime. The k^{th} dipole is located within the i^{th} basis box, and the electric field is observed at the l^{th} test point within the j^{th} test box. Position vectors $\bar{r}_i$ and $\bar{r}_j$ denote the centers of the boxes, while $\bar{w}_{ij}$ represents the translation vector between them. Local dipole positions $\bar{r}_{dk}$ and $\bar{r}_{dl}$ are measured with respect to their corresponding box centers. 55

4.5 Red arrows represent virtual dipoles distributed on the spherical surface, acquired after translation. Green arrows indicate the resulting electric field samples computed inside the test box. This setup illustrates the inverse mapping process, in which far-zone dipole representations are back-projected onto local field samples. 58

4.6 Visualization of a densely meshed RWG structure embedded in a unit cube. The triangular mesh is shown in red, with vertices and edges indicating the connectivity of the surface discretization. Although the box appears to span the interval $[-0.5, 0.5]$ along each axis in normalized coordinates, these values correspond to physical dimensions of $[-0.5\lambda, 0.5\lambda]$, resulting in an actual box size of $\lambda \times \lambda \times \lambda$. 61

4.7 Execution time comparison for computing dipole coefficients using surface and volumetric methods for varying n_d. (a) Linear scale, (b) Semilog scale. Box size is $\lambda = 1.2$ m. 62

4.8 Visualization of the RWG basis function location for different test scenarios: (a) The RWG function is located at the geometric center of the box, (b) placed near one of the interior surfaces of the box, and (c) positioned close to one of the interior corners. The box coordinates are shown in normalized units ranging from -0.5 to 0.5 along each axis, corresponding to physical dimensions of $[-0.5\lambda, 0.5\lambda]$ in each direction. 63

4.9 Mapping error comparison for volumetric and surface-based methods. (a) Error across increasing dipole resolutions ($n_d = 4$ to 9). (b) Error across varying box sizes ($\lambda/4$ to 2λ) with fixed $n_d = 4$. Results are shown for three RWG placements. 65

4.10 Visual representation of the 1000 uniformly placed test points used for evaluating spatial error distribution. 67

4.11 Colormap-based visualization of translation error across the box volume: (a) colormapped scatter of 1000 uniform points, (b) volumetric rendering of the error field. 68

4.12 Mesh representation of the test sphere using uniform triangulation. The geometry is discretized with RWG basis functions having an average edge length of 12 mm. 69

4.13 Electric field magnitude comparison for a PEC sphere of size $\frac{\lambda}{2}$, discretized using RWG basis functions with an edge length of $\frac{\lambda}{100}$. 70

4.14 Electric field magnitude comparison for a PEC sphere of size λ, discretized using RWG basis functions with an edge length of $\frac{\lambda}{50}$. . 71

4.15 Electric field magnitude comparison for a PEC sphere of size 2λ, discretized using RWG basis functions with an edge length of $\frac{\lambda}{25}$. . 72

4.16 Electric field magnitude comparison for a PEC flamme-shaped object of size 2λ, discretized using RWG basis functions with an edge length of $\frac{\lambda}{80}$. 73

4.17 Surface current magnitude distributions for the non-symmetric geometry, observed from the top view. 75

List of Tables

3.1 `constructtree()` timing results for the sphere geometry across different platforms. 33

3.2 Timing results of the `constructtree()` function for the Flamme geometry across different platforms. 35

3.3 `constructnearfield()` timing results for the sphere geometry . . 37

3.4 Timing results for GMRES solver using `mlfma()` as matrix-vector operator (38 iterations) 38

3.5 Runtime of `constructnearfield()` with different numbers of CPU cores 39

4.1 Statistical summary of translation error values computed over 1000 uniformly distributed test points. 67

A.1 Execution time values in seconds corresponding to the timing test with a densely populated box containing RWG functions, as illustrated in Fig. 4.6. 88

A.2 Error values corresponding to the accuracy test where the RWG function is located at the center of the box, as illustrated in Fig. 4.8(a). 89

A.3 Error values corresponding to the accuracy test where the RWG function is located at the surface of the box, as illustrated in Fig. 4.8(b). 89

A.4 Error values corresponding to the accuracy test where the RWG function is located at the corner of the box, as illustrated in Fig. 4.8(c). 90

Chapter 1

Introduction

Understanding and predicting the behavior of electromagnetic (EM) fields is fundamental to the design and operation of countless state-of-the-art technologies. As analytical solutions are often infeasible for realistic electromagnetic problems involving complex geometries and large domains, computational electromagnetics (CEM) has become indispensable. Leveraging modern algorithms and computing capabilities, CEM enables efficient and accurate simulation of EM behavior in a wide range of applications [1].

Among the many applications of CEM, electromagnetic scattering problems constitute a central and widely studied class [2]. These problems involve the interaction of incident electromagnetic waves, typically modeled as plane waves, with objects of arbitrary shape and material composition [3]. The goal is to determine the resulting scattered fields, especially in the far-field region, based on the geometry and electromagnetic properties of the scatterer and the surrounding medium [3]. Scattering problems are particularly challenging due to the open-domain nature of the environment and the complex boundary conditions that arise. As such, they serve as canonical benchmarks for evaluating the accuracy and efficiency of numerical solvers in CEM [1]. Their practical relevance spans a wide range of applications including radar cross-section analysis, antenna performance evaluation, wireless propagation modeling, and remote sensing [4].

For solving such open-region scattering problems efficiently, surface integral

equation (SIE) methods are particularly well-suited [2, 5, 6]. By formulating Maxwell's equations in terms of equivalent surface currents and utilizing the free-space Green's function, SIEs reduce the problem domain to the boundary of the scatterer, eliminating the need to discretize the entire surrounding space [2, 5]. This dimensionality reduction results in substantial savings in computational cost, especially when dealing with electrically large objects embedded in homogeneous media [7]. Depending on the specific nature of the problem, different formulations such as the Electric Field Integral Equation (EFIE), Magnetic Field Integral Equation (MFIE), or their combination, namely Combined Field Integral Equation (CFIE) are employed [5]. These methods provide both physical insight and numerical efficiency in modeling complex scattering phenomena [7].

To numerically solve surface integral equations arising in electromagnetic scattering problems, the Method of Moments (MoM) is one of the most widely used and well-established techniques [2, 5, 8]. In this approach, the unknown surface current distributions are represented as a linear combination of basis functions defined over a discretized mesh covering the surface of the scatterer [2, 5]. The integral equations are then tested using a set of weighting (or testing) functions, typically chosen to be the same as the basis functions in what is known as the Galerkin method [2]. This procedure results in a system of linear equations that relates the unknown coefficients of the surface currents to the excitation fields. The resulting matrix system encapsulates the electromagnetic interactions among all discretized surface elements and forms the foundation for full-wave electromagnetic simulations [9]. Due to its rigorous formulation and ability to capture detailed physical behavior, MoM has become a standard method in computational electromagnetics, particularly in the analysis of open-region scattering problems [9].

Despite its accuracy and rigorous theoretical foundation, the MoM encounters serious computational limitations when applied to electrically (with respect to wavelength) large problems [2, 5, 10]. The core challenge arises from the long-range nature of the Green's function in SIEs, which requires accounting for interactions between all pairs of surface elements. As a result, the impedance matrix becomes fully populated, leading to high computational and memory demands [2]. Specifically, storing the matrix requires $\mathcal{O}(N^2)$ memory, and solving

the system with direct solvers involves $\mathcal{O}(N^3)$ operations, where N is the number of unknowns [2, 5]. Even iterative methods, which avoid direct inversion, require a matrix-vector multiplication (MVM) at each step, costing $\mathcal{O}(N^2)$ operations per iteration. These computational burdens become prohibitive when modeling electrically large geometries, applying fine mesh discretizations, or analyzing multi-body configurations, thereby necessitating the development of more efficient algorithms that reduce both memory consumption and arithmetic complexity.

To overcome the computational and memory limitations of conventional MoM implementations, a variety of fast algorithms have been developed over the past few decades [7]. These techniques aim to accelerate MVM operations required by iterative solvers, which are typically the most time-consuming part of the overall solution process. Among these methods, some approaches exploit the localized nature of interactions or use directional basis functions to sparsify the system matrix, while others rely on the mathematical structure of the Green's function to approximate far-field interactions more efficiently [11]. Prominent examples include the Adaptive Integral Method (AIM) [12], the Precorrected Fast Fourier Transform (pFFT) [13, 14], and the Fast Multipole Method (FMM) [15, 16]. Among them, the Multilevel Fast Multipole Algorithm (MLFMA) stands out as one of the most effective and widely adopted fast solvers, particularly for large-scale three-dimensional scattering problems [17–21]. By organizing interactions hierarchically and employing analytical expansions of the Green's function, MLFMA dramatically reduces the computational complexity while preserving the accuracy of the original MoM formulation [17].

To illustrate the efficiency improvements offered by fast algorithms, it is instructive to compare their computational complexities with that of the conventional MoM. The following list summarizes the asymptotic behavior of memory and computational costs for different approaches:

- **Method of Moments (MoM):**
  Memory: $\mathcal{O}(N^2)$, Computation (direct): $\mathcal{O}(N^3)$, Computation (iterative): $\mathcal{O}(N^2)$ per iteration [2].
- **Fast Multipole Method (FMM):**

Memory: $\mathcal{O}(N^{1.5})$, Computation: $\mathcal{O}(N^{1.5})$ [11, 16].

- **Multilevel Fast Multipole Algorithm (MLFMA):**
  Memory: $\mathcal{O}(N \log N)$, Computation: $\mathcal{O}(N \log N)$ [17, 20].

Despite its favorable asymptotic complexity and widespread use, MLFMA is not without its limitations. One major challenge arises from its hierarchical structure, which introduces significant difficulties in achieving efficient parallelization [10, 17, 18, 21, 22]. In MLFMA, the computational workload is distributed unevenly across different levels of the tree and across spatial regions with varying geometry density [17, 21]. As a result, balancing the load among processing units becomes nontrivial, often requiring sophisticated partitioning strategies and communication schemes. In practice, this leads to suboptimal scalability for parallelization, particularly in distributed memory systems where communication overhead can dominate the overall computation time.

Another well-known limitation of MLFMA is the low-frequency breakdown (LFB) problem. The core of MLFMA relies on multipole and plane wave expansions of the Green's function, which are asymptotically valid in the high-frequency regime [7]. However, when the box sizes become electrically small, i.e., smaller than the wavelength, these expansions begin to lose accuracy [23]. This leads to numerical instabilities and degraded solution quality at low frequencies or in multi-scale problems where small and large geometrical features coexist. Although several remedies have been proposed in the literature to address LFB, such as using alternative Green's function expansions [24], hybrid formulations [25], or higher-precision arithmetic [19, 26], these approaches often require substantial changes to the MLFMA framework and may increase implementation complexity or computational cost [10].

These limitations motivate the search for alternative strategies that can retain the favorable complexity of MLFMA while improving parallelization and maintaining robustness across a broader frequency range [27, 28].

A recent approach proposed in [29] introduces a novel method that addresses both the parallelization and low-frequency limitations of MLFMA by replacing its hierarchical structure with a group-wise interaction model based on uniformly distributed volumetric dipoles and machine learning [6,30]. In this framework, the

scatterer geometry is partitioned into equal-sized boxes, and the local subdomain basis functions within each box are represented using a fixed number of arbitrarily oriented Hertzian dipoles distributed uniformly throughout the box volume. Far-field interactions between boxes are then computed using machine-learned models trained to predict dipole-to-dipole interactions based on the free-space Green's function [29]. Because each box uses the same number and configuration of dipoles, the interaction workload becomes identical across the entire geometry, enabling strong scalability in parallel implementations [31, 32]. Furthermore, by training the models directly on full-wave dipole interactions, the method is shown to be inherently immune to the LFB problem [6, 29].

The resulting technique achieves accurate and efficient MVMs for surface and volume integral equations, offering a viable alternative to traditional fast solvers. Moreover, since the neural networks are trained offline and are geometry-independent (as long as box sizes and relative distances between the boxes are preserved), the method can be reused across different problems without retraining [29]. Numerical results demonstrate the approach's effectiveness on a range of scattering problems, confirming its robustness and strong scalability feature in parallelization even in electrically small or multi-scale regimes.

Building on the core ideas of this volumetric dipole-based framework, in this thesis we propose a key modification: instead of placing the virtual Hertzian dipoles throughout the volume of each box, we distribute them on the surface of each box. This change is motivated by two main reasons. First, from an electromagnetic perspective, it is well-established through the uniqueness theorem that the radiated field outside a region can be fully represented by equivalent sources on its enclosing surface [3]. This suggests that, for far-field interactions, surface-based representations can achieve comparable accuracy to volumetric ones. Second, and more importantly from a computational standpoint, the number of required dipoles grows significantly more slowly when confined to the surface. While a volumetric grid with resolution n in each direction results in n^3 dipoles per box, a surface-based grid requires only $6n^2$ dipoles. This reduction becomes especially beneficial for electrically large problems, where large boxes are required and volumetric sampling becomes increasingly inefficient.

By adopting a surface-based dipole placement strategy, the proposed method

aims to improve both memory efficiency and computational cost, while preserving the main advantages of the original framework, namely, strong parallel scalability, robustness against LFB, and geometry independence. Similar to established fast solvers such as the Adaptive Integral Method (AIM) [33] and the Discrete Dipole Approximation (DDA) [34], the proposed approach maps the problem onto a fixed set of dipole basis functions—in this case, uniformly distributed surface dipoles per box. This fixed mapping enforces a constant number of degrees of freedom across the entire computational domain, independent of the original discretization density, which, from a machine learning perspective, is particularly advantageous as it constrains the learning problem to a well-defined, limited feature space. This fixed-dimensional representation enables more stable and efficient training of neural networks to approximate dipole-to-dipole interactions. In the current implementation, however, these interactions are computed directly from the Green's function without incorporating any learning-based approximations. Simulation results obtained in this setting demonstrate that the surface-based formulation can provide accurate solutions with significantly fewer dipoles per box and reduced computational overhead. These findings indicate that integrating machine learning in future work may further enhance both efficiency and generalization capabilities.

In addition to these benefits, efficient parallelization strategies are essential for tackling electrically large problems in practice. In computational electromagnetics, two main paradigms are commonly used for parallelization: Central Processing Units (CPU) based and Graphics Processing Unit (GPU) based computing [28, 35]. Multi-core CPUs, often programmed using shared-memory models such as OpenMP or high-level tools like Numba, allow moderate acceleration by distributing the workload across processor cores [31, 36]. GPUs, on the other hand, provide massive parallelism through thousands of lightweight threads, making them highly effective for compute-intensive tasks such as MVMs, near-field evaluations, and field propagation [37]. Despite their advantages, both CPU and GPU approaches face challenges when applied to hierarchical algorithms like MLFMA, which require careful scheduling, memory management, and load balancing to achieve optimal performance [27, 37].

In contrast to hierarchical algorithms such as MLFMA, the proposed dipole-based framework is theoretically more suitable for parallel execution [6, 29]. Since the scatterer is divided into uniformly sized boxes and each box employs the same number and configuration of dipoles, the computational workload is expected to be naturally balanced across the domain.Far-field interactions between boxes are independent, simplifying potential parallel implementations. These properties are anticipated to eliminate the need for level-wise traversal or complex data dependencies, potentially enabling straightforward implementation on both multi-core CPUs and massively parallel GPUs. Although a parallel implementation of the surface-based method has not yet been realized in this study, its structural characteristics indicate strong potential for effective parallelization in future work.

To complement the theoretical analysis of the proposed framework's parallelization potential, we also developed a prototype MLFMA implementation using Python and Numba [36]. This implementation focuses on parallelizing the most computationally intensive stages of the algorithm, namely the translation and aggregation steps, using shared-memory parallelism on modern multi-core CPUs. Leveraging Numba's Just-In-Time (JIT) compilation and automatic thread-level parallelism, the implementation allows rapid development without the need for low-level parallel programming constructs [38]. Although this CPU-based approach does not fully address MLFMA's inherent load imbalance issues, it provides valuable insight into the bottlenecks of hierarchical methods and demonstrates that even modest parallelization can yield significant speedups in practical settings. These results also serve as a useful baseline for evaluating the performance and scalability for parallelization of the proposed non-hierarchical dipole-based method [39].

While the current parallelization efforts focus on shared-memory CPU architectures, further acceleration could be achieved by offloading MLFMA computations to GPUs using Compute Unified Device Architecture (CUDA) [37]. The massively parallel nature of GPUs makes them particularly well-suited for the large number of independent operations involved in translation and aggregation steps [40]. However, adapting MLFMA to GPU architectures introduces additional challenges, such as memory coalescing, kernel optimization, and efficient data transfer between host and device [27]. Despite these complexities, recent

advances in GPU programming and the availability of mature CUDA libraries make this a promising direction for future work. A successful GPU implementation could significantly reduce runtime and enable large-scale simulations that are currently limited by CPU-bound performance.

Although this thesis presents significant advancements by proposing a surface-based dipole interaction framework and developing a parallel MLFMA prototype, an important long-term objective is to integrate these approaches into a unified, fully GPU-accelerated solver. Such integration is expected to combine the computational efficiency and parallelization capability of the new surface-based method with the established accuracy of hierarchical algorithms like MLFMA, thereby enabling extremely large-scale electromagnetic simulations. In addition to parallelizing the surface-based method, it may be further accelerated by incorporating advanced machine learning techniques. Achieving this will require addressing remaining challenges related to GPU implementation, load balancing, and multi-scale robustness, which represent key challenges for future exploration in computational electromagnetics.

In summary, this thesis presents two key contributions to the field of computational electromagnetics. First, to better understand and benchmark the performance of hierarchical solvers, a parallel MLFMA implementation is developed using Python and Numba. This CPU-based prototype highlights the bottlenecks in conventional MLFMA workflows and serves as a reference point for evaluating the performance potential of alternative methods. Second, a surface-based dipole interaction framework is introduced as an alternative to traditional hierarchical solvers, extending a previously proposed volumetric approach [29]. By distributing Hertzian dipoles uniformly on the surfaces of discretized subdomains and aiming to model their far-field interactions using machine-learned Green's function responses, the proposed method is designed to enable accurate, low-frequency-stable, and highly parallelizable simulations with significantly reduced computational cost.

This thesis implements the surface-based dipole interaction method using direct evaluations of the Green's function, without incorporating learning-based models or GPU acceleration. Numerical experiments were carried out to assess its accuracy and efficiency in this simplified setting. In parallel, a prototype

MLFMA implementation was developed to investigate the performance characteristics of hierarchical methods. This implementation leverages Numba-based JIT compilation along with shared-memory CPU parallelization to accelerate the most computationally intensive stages of MLFMA, such as translation and aggregation. While these implementations demonstrate the feasibility and performance benefits of the proposed strategies, several components remain as future work. In particular, parallelizing the surface-based method, integrating machine-learned dipole interactions, and developing a unified GPU-accelerated solver constitute important directions for future research.

The remainder of this thesis is organized as follows: Chapter 2 provides a comprehensive overview of integral equation formulations in electromagnetics, focusing on MoM and MLFMA. It also introduces key computational optimizations such as JIT compilation and CPU-based parallelization in Python that are necessary during the numerical implementations of MoM and MLFMA. Chapter 3 presents detailed timing analyses and performance comparisons of these implementations across different environments (MATLAB and Python), emphasizing the impact of code-level acceleration techniques. Chapter 4 introduces a surface-distributed dipole modeling framework as the foundation for a future machine-learning-based solver, and presents preliminary numerical results obtained using direct evaluations of the Green's function, benchmarked against MLFMA. Chapter 5 concludes the thesis by summarizing the main findings and outlining potential future directions for developing fast and scalable solvers for electromagnetic scattering problems. Throughout this thesis, an implicit $e^{j\omega t}$ time convention is used and suppressed.

Chapter 2

Preliminaries

2.1 Electromagnetic related preliminaries

This section provides a brief overview of the fundamental electromagnetic theory that forms the basis of the numerical methods discussed in this thesis. Key equations, field relationships, and physical concepts relevant to wave propagation, radiation, and scattering are introduced. These preliminaries serve to establish a common framework for interpreting and implementing computational techniques such as MoM and MLFMA, which are presented in later chapters.

2.1.1 Basics of Electromagnetic Theory

In a homogeneous, linear, and isotropic medium with angular frequency ω, and material properties μ and ε, Maxwell's equations in the frequency (phasor) domain can be expressed in differential form as follows:

$$\nabla \times \bar{E}(\bar{r}) = -j\omega\mu\bar{H}(\bar{r}) - \bar{M}(\bar{r}), \tag{2.1}$$

$$\nabla \times \bar{H}(\bar{r}) = j\omega\epsilon\bar{E}(\bar{r}) + \bar{J}(\bar{r}), \tag{2.2}$$

$$\nabla \cdot \bar{D}(\bar{r}) = \rho_e(\bar{r}), \tag{2.3}$$

$$\nabla \cdot \bar{B}(\bar{r}) = \rho_m(\bar{r}). \tag{2.4}$$

In addition to Maxwell's equations, the conservation of charges in the frequency domain is described by the continuity equations given by

$$\nabla \cdot \bar{J}(\bar{r}) = -j\omega\rho_e(\bar{r}), \tag{2.5}$$

$$\nabla \cdot \bar{M}(\bar{r}) = -j\omega\rho_m(\bar{r}). \tag{2.6}$$

In (2.1)–(2.6), $\bar{E}(\bar{r})$ and $\bar{H}(\bar{r})$ denote the electric and magnetic field intensities (in V/m and A/m, respectively), while $\bar{D}(\bar{r})$ and $\bar{B}(\bar{r})$ represent the electric and magnetic flux densities (in C/m^2 and T). These quantities are related by the constitutive relations $\bar{D} = \varepsilon\bar{E}$ and $\bar{B} = \mu\bar{H}$, where ε and μ are the permittivity and permeability of the medium, respectively. The terms $\rho_e(\bar{r})$ and $\rho_m(\bar{r})$ correspond to the electric and the fictitious magnetic charge densities respectively, measured in C/m^3 and Wb/m^3. Similarly, $\bar{J}(\bar{r})$ and $\bar{M}(\bar{r})$ correspond to the electric and the fictitious magnetic current densities, respectively, measured in A/m^2 and C/m^2. Taking the curl of (2.1), we obtain the following vector wave equation for $\bar{E}(\bar{r})$:

$$\nabla \times \nabla \times \bar{E}(\bar{r}) = -j\omega\mu\nabla \times \bar{H}(\bar{r}) - \nabla \times \bar{M}(\bar{r}). \tag{2.7}$$

Then, by applying the vector identity $\nabla \times \nabla \times \bar{A} = \nabla(\nabla \cdot \bar{A}) - \nabla^2\bar{A}$ to (2.7), we obtain:

$$\nabla\nabla \cdot \bar{E}(\bar{r}) - \nabla^2\bar{E}(\bar{r}) = -j\omega\mu\left[j\omega\epsilon\bar{E}(\bar{r}) + \bar{J}(\bar{r})\right] - \nabla \times \bar{M}(\bar{r}), \tag{2.8}$$

$$\frac{1}{\epsilon}\nabla\rho_e(\bar{r}) - \nabla^2\bar{E}(\bar{r}) = \omega^2\mu\epsilon\bar{E}(\bar{r}) - j\omega\mu\bar{J}(\bar{r}) - \nabla \times \bar{M}(\bar{r}), \tag{2.9}$$

$$\nabla^2\bar{E}(\bar{r}) + k^2\bar{E}(\bar{r}) = j\omega\mu\bar{J}(\bar{r}) + \frac{1}{\epsilon}\nabla\rho_e(\bar{r}) + \nabla \times \bar{M}(\bar{r}). \tag{2.10}$$

Similarly, by applying the same procedure to (2.2), we obtain the Helmholtz equation for $\bar{H}(\bar{r})$ as:

$$\nabla^2\bar{H}(\bar{r}) + k^2\bar{H}(\bar{r}) = j\omega\mu\bar{M}(\bar{r}) + \frac{1}{\epsilon}\nabla\rho_m(\bar{r}) - \nabla\times\bar{J}(\bar{r}). \tag{2.11}$$

In addition, assuming a homogeneous medium, the electric and magnetic fields can be expressed in terms of scalar and vector potentials as follows [18]:

$$\bar{E}(\bar{r}) = -j\omega\bar{F}(\bar{r}) - \nabla\phi_e(\bar{r}) - \frac{1}{\epsilon}\nabla\times\bar{A}(\bar{r}), \tag{2.12}$$

$$\bar{H}(\bar{r}) = -j\omega\bar{A}(\bar{r}) - \nabla\phi_m(\bar{r}) + \frac{1}{\mu}\nabla\times\bar{F}(\bar{r}). \tag{2.13}$$

In (2.12)–(2.13), $\bar{A}(\bar{r})$ and $\bar{F}(\bar{r})$ denote the electric and magnetic vector potentials, respectively, while ϕ_e and ϕ_m represent the corresponding electric and magnetic scalar potentials. To proceed, we impose the Lorenz gauge conditions given by

$$\nabla\cdot\bar{A}_m(\bar{r}) = -j\omega\epsilon\mu\,\phi_e(\bar{r}), \tag{2.14}$$

$$\nabla\cdot\bar{A}_e(\bar{r}) = -j\omega\mu\epsilon\,\phi_m(\bar{r}). \tag{2.15}$$

Finally, by using the Maxwell equations (2.1)–(2.4) and the potential relations in (2.12)–(2.15), we derive the Helmholtz equations for the scalar and vector potentials as follows:

$$\nabla^2\phi_e(\bar{r}) + k^2\phi_e(\bar{r}) = -\frac{1}{\epsilon}\rho_e(\bar{r}), \tag{2.16}$$

$$\nabla^2\phi_m(\bar{r}) + k^2\phi_m(\bar{r}) = -\frac{1}{\mu}\rho_m(\bar{r}), \tag{2.17}$$

$$\nabla^2\bar{A}(\bar{r}) + k^2\bar{A}(\bar{r}) = -\epsilon\bar{M}(\bar{r}), \tag{2.18}$$

$$\nabla^2\bar{F}(\bar{r}) + k^2\bar{F}(\bar{r}) = -\mu\bar{J}(\bar{r}). \tag{2.19}$$

2.1.2 Green's Function

To solve the Helmholtz equations presented in (2.16)–(2.19), we make use of the Green's function technique. For this purpose, we employ the homogeneous-space Green's function given by

$$g(\bar{r},\bar{r}') = \frac{e^{ik|\bar{r}-\bar{r}'|}}{4\pi|\bar{r}-\bar{r}'|}, \tag{2.20}$$

which represents the solution of the scalar Helmholtz equation at an observation point $\bar{r}$ for a unit point source located at $\bar{r}'$, and it forms the fundamental solution for constructing more general field expressions through integration. By using the Green's function $g(\bar{r}, \bar{r}')$ as the kernel, the scalar and vector potentials corresponding to arbitrary source distributions can be written as

$$\phi_e(\bar{r}) = \frac{1}{\epsilon} \int g(\bar{r}, \bar{r}')\, \rho_e(\bar{r}') d\bar{r}', \tag{2.21}$$

$$\phi_m(\bar{r}) = \frac{1}{\mu} \int g(\bar{r}, \bar{r}')\, \rho_m(\bar{r}') d\bar{r}', \tag{2.22}$$

$$\bar{A}(\bar{r}) = \epsilon \int g(\bar{r}, \bar{r}')\, \bar{M}(\bar{r}') d\bar{r}', \tag{2.23}$$

$$\bar{F}(\bar{r}) = \mu \int g(\bar{r}, \bar{r}')\, \bar{J}(\bar{r}') d\bar{r}'. \tag{2.24}$$

$$\tag{2.25}$$

By substituting the expressions for the scalar and vector potentials given in equations (2.21)–(2.24) into the potential-field relations defined in (2.12) and (2.13), the resulting electric and magnetic fields can be expressed as follows:

$$\begin{aligned} \bar{E}(\bar{r}) = -jk\eta \int \left[\bar{J}(\bar{r}') + \frac{1}{k^2} \nabla' \cdot \bar{J}(\bar{r}') \nabla \right] g(\bar{r}, \bar{r}') d\bar{r}' \\ - \int \nabla g(\bar{r}, \bar{r}') \times \bar{M}(\bar{r}') d\bar{r}', \end{aligned} \tag{2.26}$$

$$\begin{aligned} \bar{H}(\bar{r}) = -\frac{jk}{\eta} \int \left[\bar{M}(\bar{r}') + \frac{1}{k^2} \nabla' \cdot \bar{M}(\bar{r}') \nabla \right] g(\bar{r}, \bar{r}') d\bar{r}' \\ + \int \nabla g(\bar{r}, \bar{r}') \times \bar{J}(\bar{r}') d\bar{r}'. \end{aligned} \tag{2.27}$$

2.1.3 Surface Equivalance Principle

The surface equivalence theorem provides a powerful method to simplify electromagnetic scattering problems by replacing complex materials or internal fields with equivalent surface sources. As shown in Fig. 2.1, the original configuration on the left involves a perfectly electrically conducting (PEC) object located in

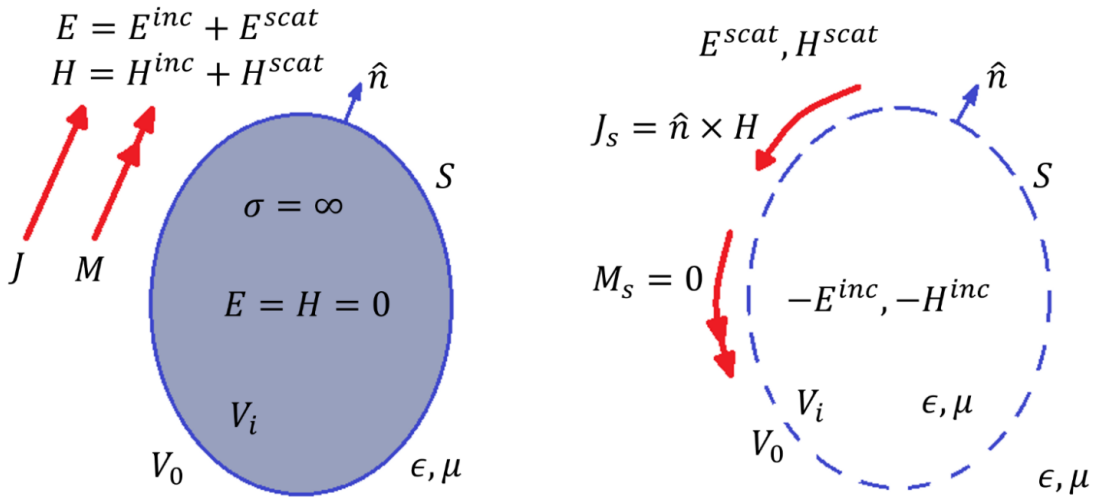


Figure 2.1: Illustration of the surface equivalence theorem applied to a perfectly conducting object.

region V_i with interior fields $\bar{E} = \bar{H} = 0$, and immersed in a homogeneous background medium (ϵ, μ) in region V_0. The total fields in the exterior region are expressed as the sum of the incident and scattered fields as

$$\bar{E} = \bar{E}^{\text{inc}} + \bar{E}^{\text{scat}}, \quad \bar{H} = \bar{H}^{\text{inc}} + \bar{H}^{\text{scat}}.$$

According to the equivalence principle, the physical object can be removed and replaced with equivalent surface currents placed on its boundary, such that the fields in the exterior region remain unchanged. These equivalent currents are defined as

$$\bar{J}_s = \hat{n} \times \bar{H}, \tag{2.28}$$

$$\bar{M}_s = -\hat{n} \times \bar{E}. \tag{2.29}$$

On the right side of Fig. 2.1 (In this specific case, $\bar{M}_s = 0$ not because of the absence of magnetic sources, but due to the PEC boundary condition $\bar{E} = 0$ on the surface, which implies $\bar{M}_s = -\hat{n} \times \bar{E} = 0$.), the PEC object has been replaced with an equivalent problem: the region V_i is now filled with the same homogeneous medium as V_0, and equivalent surface currents radiate the same fields as in the original configuration. The scattered fields are preserved in V_0,

and the fields in V_i are constructed as the negatives of the incident fields (i.e., $-\bar{E}^{\text{inc}}, -\bar{H}^{\text{inc}}$) to satisfy the PEC boundary conditions. This transformation allows us to analyze scattering from complex structures using only surface integrals, leading to significant simplification in both analytical and numerical formulations.

2.1.4 Electric Field Integral Equations (EFIE)

In order to derive EFIEs, it is essential to first examine the general structure of the electric field in the presence of a scattering object. When an electromagnetic wave interacts with such an object, the total electric field at a point in space can be represented as the superposition of the incident and scattered fields given as

$$\bar{E}_{\text{total}}(\bar{r}) = \bar{E}_{\text{inc}}(\bar{r}) + \bar{E}_{\text{scat}}(\bar{r}). \tag{2.30}$$

In this representation, $\bar{E}_{\text{inc}}(\bar{r})$ denotes the electric field generated by external sources, while $\bar{E}_{\text{scat}}(\bar{r})$ represents the field scattered by the object as a result of its interaction with the incident wave. As previously discussed, the scattered field can be modeled using the surface equivalence theorem. Specifically, it is the electromagnetic field radiated by the equivalent surface currents $\bar{J}_s$ and $\bar{M}_s$, which are introduced on the boundary of the object in accordance with the equivalence principle.

The integral expressions used to compute the scattered fields are, in fact, consistent with the general field formulations given earlier in (2.26) and (2.27). However, in the specific case considered here, where the object is assumed to be a PEC, only the electric surface current density $\bar{J}(\bar{r}')$ contributes to the scattered field. As a result, the expression for the scattered electric field reduces to a modified form of (2.26), given by

$$\bar{E}^{\text{scat}}(\bar{r}) = -j\omega\mu \int \left(\bar{J}(\bar{r}') + \frac{1}{k^2} \nabla' \cdot \bar{J}(\bar{r}') \nabla \right) g(\bar{r}, \bar{r}')\, d\bar{r}'. \tag{2.31}$$

In the context of PEC objects, the tangential component of the total electric field on the surface must vanish to satisfy the boundary condition. That is,

$$\hat{n} \times \bar{E}_{\text{total}}(\bar{r}) = \hat{n} \times \left(\bar{E}_{\text{inc}}(\bar{r}) + \bar{E}_{\text{scat}}(\bar{r}) \right) = 0, \quad \bar{r} \in S, \tag{2.32}$$

where $\hat{n}$ denotes the unit outward normal vector on the surface S as shown in Fig 2.1. Rearranging this condition leads to the classical form of EFIE:

$$\hat{n} \times \bar{E}_{\text{scat}}(\bar{r}) = -\hat{n} \times \bar{E}_{\text{inc}}(\bar{r}). \tag{2.33}$$

This form is commonly referred to as the *normal form of EFIE* (N-EFIE) [18], as it arises directly from enforcing the vanishing of the tangential component via a single cross product with the normal vector. However, in numerical implementations, it is often advantageous to reformulate this into a fully tangential form. This is achieved by applying a second cross product with the normal vector, as given by

$$\hat{n} \times \hat{n} \times \bar{E}_{\text{scat}}(\bar{r}) = -\hat{n} \times \hat{n} \times \bar{E}_{\text{inc}}(\bar{r}), \tag{2.34}$$

which isolates the tangential components more robustly. The formulation in (2.34) is known as the *tangential EFIE* (T-EFIE) [18]. It ensures compatibility with commonly used basis functions such as Rao-Wilton-Glisson (RWG) functions [41], which are defined over the surface of the scatterer and naturally conform to the tangential vector field space. This formulation is therefore widely adopted in surface integral equation solvers for PEC scattering problems.

2.1.5 Discretization via the MoM

EFIE introduced earlier is an operator equation that cannot be solved analytically in most practical scenarios. Therefore, a numerical discretization technique is required. One of the most widely used methods for this purpose is MoM, which converts the continuous operator equation into a system of linear equations.

Consider a generic linear operator equation of the form

$$\mathcal{L}\{\bar{f}(\bar{r})\} = \bar{g}(\bar{r}), \tag{2.35}$$

where $\mathcal{L}$ is a linear operator, $\bar{f}(\bar{r})$ is the unknown vector function to be determined, and $\bar{g}(\bar{r})$ is a known excitation function. In MoM, the unknown function

$\bar{f}(\bar{r})$ is expanded as a linear combination of a finite set of known basis functions $\bar{b}_n(\bar{r})$ as

$$\bar{f}(\bar{r}) \approx \sum_{n=1}^{N} a[n]\bar{b}_n(\bar{r}), \tag{2.36}$$

where $a[n]$ are the unknown coefficients to be computed.

Substituting this expansion into the operator equation (2.35) yields

$$\sum_{n=1}^{N} a[n]\mathcal{L}\{\bar{b}_n(\bar{r})\} = \bar{g}(\bar{r}). \tag{2.37}$$

To solve this equation, we project both sides onto a set of testing (or weighting) functions $\bar{t}_m(\bar{r})$ for $m = 1, 2, \ldots, N$, yielding

$$\int \bar{t}_m(\bar{r}) \cdot \sum_{n=1}^{N} a[n]\mathcal{L}\{\bar{b}_n(\bar{r})\}d\bar{r} = \int \bar{t}_m(\bar{r}) \cdot \bar{g}(\bar{r})d\bar{r}, \quad (m = 1, 2, \ldots, N). \tag{2.38}$$

Interchanging the order of summation and integration leads to

$$\sum_{n=1}^{N} a[n] \int \bar{t}_m(\bar{r}) \cdot \mathcal{L}\{\bar{b}_n(\bar{r})\}d\bar{r} = \int \bar{t}_m(\bar{r}) \cdot \bar{g}(\bar{r})d\bar{r}, \quad (m = 1, 2, \ldots, N). \tag{2.39}$$

The final equation can now be expressed in matrix-vector form as

$$\sum_{n=1}^{N} a[n]\bar{\bar{Z}}[m, n] = \bar{v}[m], \quad (m = 1, 2, \ldots, N), \tag{2.40}$$

which is usually written compactly in matrix-vector notation as

$$\bar{\bar{Z}} \cdot \bar{\alpha} = \bar{v}. \tag{2.41}$$

In (2.41), the entries of the known impedance matrix $\bar{\bar{Z}}$ and the excitation vector $\bar{v}$ are defined as

$$\bar{\bar{Z}}[m, n] = \int \bar{t}_m(\bar{r}) \cdot \mathcal{L}\{\bar{b}_n(\bar{r})\}d\bar{r}, \tag{2.42}$$

$$\bar{v}[m] = \int \bar{t}_m(\bar{r}) \cdot \bar{g}(\bar{r})d\bar{r}. \tag{2.43}$$

Finally in (2.41), $\bar{\alpha}$ is the unknown current vector, whose entries are $\alpha_n = a[n]$. Once the system in (2.41) is assembled, standard linear algebra techniques can be used to solve for $\bar{\alpha}$.

2.1.6 Rao-Wilton-Glisson (RWG) Basis Functions

In the context of surface integral equation formulations for electromagnetic scattering problems, the choice of basis functions plays a crucial role in accurately representing surface current densities. For triangulated surfaces, one of the most widely adopted sets of basis functions is the RWG functions, introduced in 1982 [41]. These vector basis functions are specifically designed to approximate the current densities on the surfaces of PEC objects.

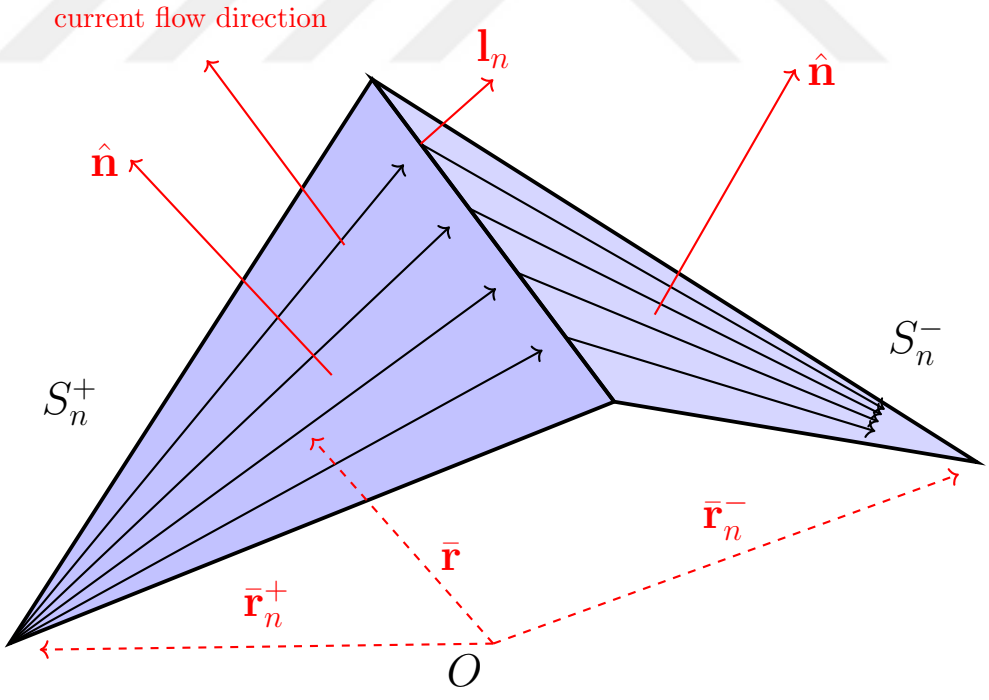


Figure 2.2: Illustration of an RWG basis function.

RWG basis functions are defined on pairs of adjacent triangular surface patches sharing a common edge, as depicted in Fig. 2.2. Let S_n^+ and S_n^- denote the two triangles sharing an edge of length l_n, with surface areas A_n^+ and A_n^-, and free (non-shared) vertices located at $\bar{r}_n^+$ and $\bar{r}_n^-$, respectively. The RWG basis function

$\bar{b}_n(\bar{r})$ associated with this edge is defined piecewise over the two triangles as:

$$\bar{b}_n^{\mathrm{RWG}}(\bar{r}) = \begin{cases} \dfrac{l_n}{2A_n^+}(\bar{r} - \bar{r}_n^+), & \bar{r} \in S_n^+ \\ \dfrac{l_n}{2A_n^-}(\bar{r}_n^- - \bar{r}), & \bar{r} \in S_n^- \\ 0, & \text{otherwise.} \end{cases} \tag{2.44}$$

The divergence of the RWG basis function is constant over each triangle and given by:

$$\nabla \cdot \bar{b}_n^{\mathrm{RWG}}(\bar{r}) = \begin{cases} \dfrac{l_n}{A_n^+}, & \bar{r} \in S_n^+ \\ -\dfrac{l_n}{A_n^-}, & \bar{r} \in S_n^- \\ 0, & \text{otherwise.} \end{cases} \tag{2.45}$$

RWG basis functions possess several advantageous properties that make them particularly suitable for surface integral equation formulations.

2.1.7 Multilevel Fast Multipole Algorithm (MLFMA)

As previously established, MoM results in a dense system of linear equations of the form

$$\bar{\bar{Z}}\bar{\alpha} = \bar{v}, \tag{2.46}$$

where $\bar{\bar{Z}}$ is a fully populated matrix, whose entries involve double integrals over basis and testing functions. As the number of unknowns N increases, especially in electrically large structures, the computational and memory costs of solving this system become prohibitively high. As explained before, direct matrix inversion methods have a complexity of $\mathcal{O}(N^3)$, and iterative solvers require $\mathcal{O}(N^2)$ complexity per MVM. The memory usage also grows as $\mathcal{O}(N^2)$, limiting the applicability of MoM to large-scale problems. To address these limitations, MLFMA is used to accelerate the MVM operation. MLFMA hierarchically organizes interactions between basis and testing functions and efficiently computes far-field contributions through a plane wave representation of the Green's function, which

is a diagonalized form of the multipole expansion. This representation decouples source and observation points, enabling efficient aggregation, translation, and disaggregation stages within a multilevel hierarchy. As a result, the per-iteration computational complexity is reduced from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$, and memory requirements are significantly decreased. Consequently, MLFMA enables the practical solution of large-scale scattering problems involving millions of unknowns.

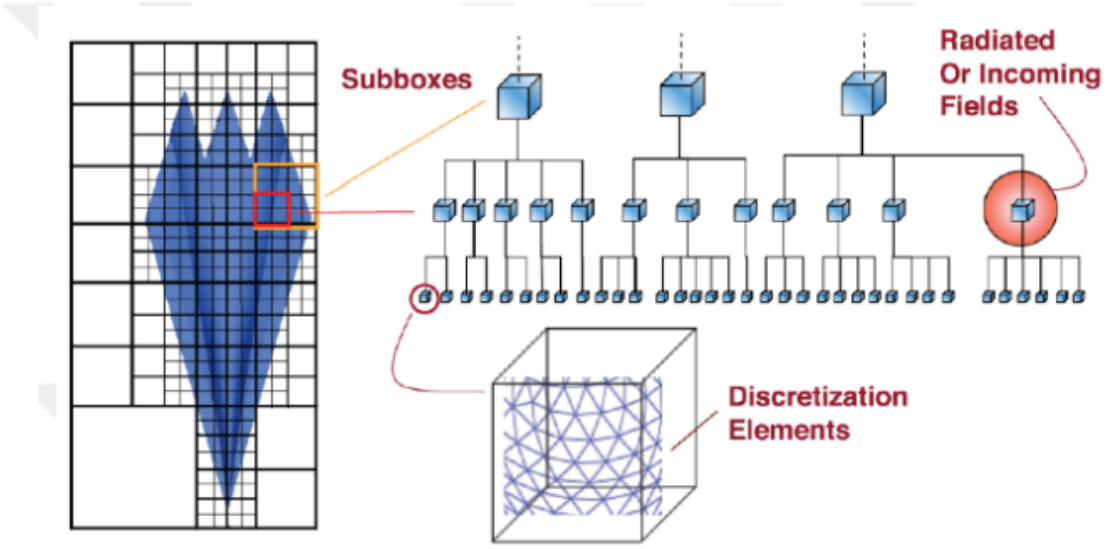


Figure 2.3: Illustration of the tree structure of MLFMA. [18]

MLFMA partitions the computational domain into a hierarchy of nested boxes. Beginning at the finest level, where each box contains a small number of elements, groups of eight adjacent boxes are recursively merged to form higher-level boxes, as illustrated by the hierarchical tree structure shown in Fig. 2.3. This spatial hierarchy enables a structured classification of electromagnetic interactions based on the distance between source and observation regions.

Leveraging this classification, the MVM operation, $\bar{\bar{Z}} \cdot \bar{\alpha} = \bar{v}$, is reformulated by decomposing the impedance matrix into near-field (NF) and far-field (FF) components given as

$$\bar{\bar{Z}}\bar{\alpha} = \bar{\bar{Z}}_{\mathrm{NF}}\bar{\alpha} + \bar{\bar{Z}}_{\mathrm{FF}}\bar{\alpha} = \bar{v}, \tag{2.47}$$

where $\bar{\bar{Z}}_{\mathrm{NF}}$ denotes the near-field impedance matrix, calculated by directly evaluating the dyadic Green's function between RWG basis pairs, and $\bar{\bar{Z}}_{\mathrm{FF}}$ represents the far-field part computed using MLFMA. This decomposition enables the selective application of computational strategies across different spatial regions, significantly accelerating the solution process.

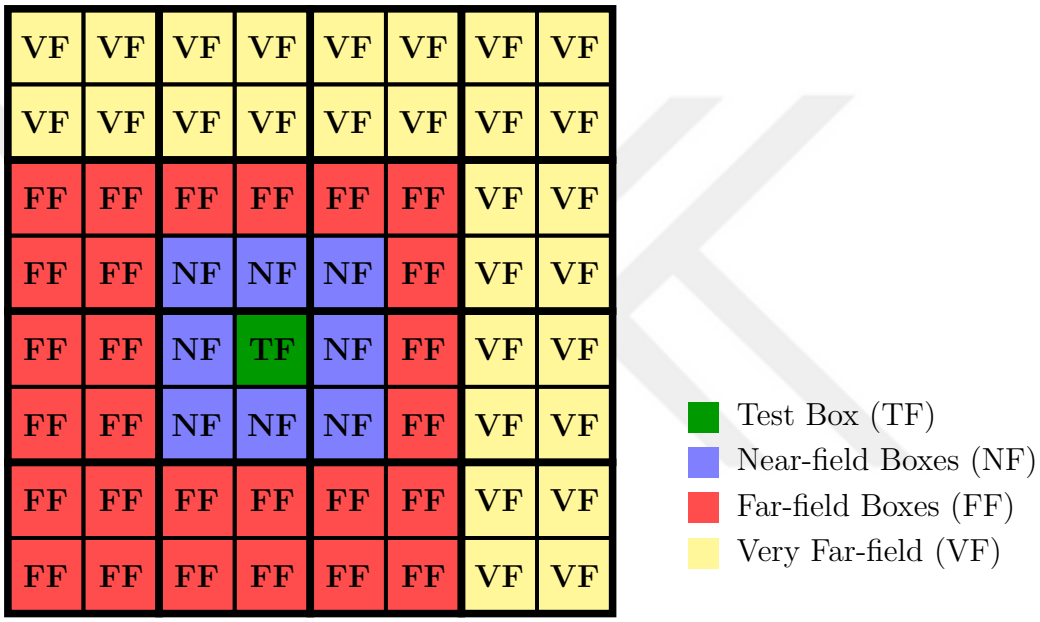


Figure 2.4: Two-dimensional illustration of the spatial classification of an 8×8 computational grid in MLFMA: the test box (green), near-field region (blue), far-field region (red), and very far-field region (yellow) are identified based on hierarchical box proximity.

As shown in Fig. 2.4 for a two-dimensional illustration, the computational domain is spatially decomposed into distinct interaction regions based on proximity. The green box indicates the test box, blue boxes represent its near-field neighbors, red boxes denote the far-field region. Finally, yellow boxes belong to the very far-field region.

The near-field interactions are computed directly through numerical evaluation of the dyadic Green's function between basis and testing RWG pairs. In contrast, the far-field contributions are approximated using a diagonalized plane-wave expansion of the Green's function. This approximation enables the use of the hierarchical aggregation, translation, and disaggregation stages of MLFMA. Through

this multilevel structure, redundant calculations are avoided and the overall cost of matrix-vector multiplication is reduced to $\mathcal{O}(N \log N)$.

In MLFMA, the far-field approximation introduces a controlled error, making the solution an approximation to the original dense system. The accuracy depends heavily on the modeling of far-field interactions, which is enabled by factorizing and diagonalizing the Green's function using Gegenbauer's addition theorem.

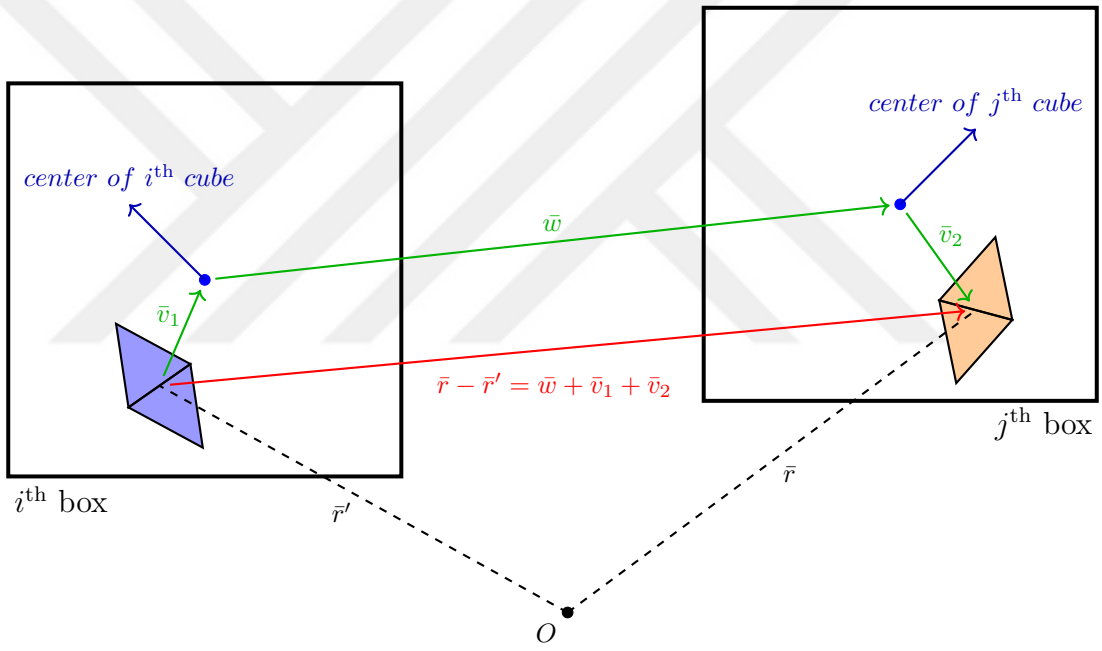


Figure 2.5: Vector decomposition between two RWGs located in the i^{th} and j^{th} far-field boxes. The vector $\bar{w}$ denotes the translation vector connecting the centers of these boxes.

Consider the interaction between a testing and a basis function, where the distance can be decomposed as

$$\bar{r} - \bar{r}' = \bar{w} + \bar{v} \tag{2.48}$$

with $\bar{w}$ denoting the vector between group centers, and $\bar{v} = \bar{v}_1 + \bar{v}_2$ representing local offsets as shown in Fig. 2.5. If $w = |\bar{w}| > |\bar{v}|$, the scalar Green's function admits the Gegenbauer expansion given by

$$\frac{e^{jk|\bar{r}-\bar{r}'|}}{4\pi|\bar{r}-\bar{r}'|} = \frac{jk}{4\pi} \sum_{t=0}^{\infty} (-1)^t (2t+1) j_t(kv) h_t^{(1)}(kw) P_t(\hat{v} \cdot \hat{w}). \tag{2.49}$$

In this expansion, $j_t(\cdot)$ and $h_t^{(1)}(\cdot)$ denote the spherical Bessel and spherical Hankel functions of the first kind, respectively, and $P_t(\cdot)$ represents the Legendre polynomial of degree t. Although (2.49) is mathematically valid, it is not directly suitable for efficient implementation in MLFMA. Therefore, the Green's function is further diagonalized using plane-wave expansions given as

$$\frac{\exp\left(jk|\bar{w}+\bar{v}_1+\bar{v}_2|\right)}{4\pi|\bar{w}+\bar{v}_1+\bar{v}_2|} \approx \frac{jk}{(4\pi)^2}\int \beta(\hat{k},\bar{v}_1)\,\alpha_\tau(\hat{k},\bar{w})\,\beta(\hat{k},\bar{v}_2)d^2\hat{k}, \tag{2.50}$$

which decouples source and observer dependencies and enables efficient implementation in MLFMA.

In the left hand side of (2.50) *shift function* is defined as

$$\beta(\bar{k},\bar{v}) = e^{j\bar{k}\cdot\bar{v}}, \tag{2.51}$$

and the *translation function* is given by

$$\alpha_\tau(\hat{k},\bar{w}) = \sum_{t=0}^{\tau}(j)^t(2t+1)h_t^{(1)}(k|\bar{w}|)P_t(\hat{k}\cdot\hat{w}). \tag{2.52}$$

The truncation number τ determines the number of terms retained in the spherical expansion. It also determines the angular resolution used in integration. It directly influences the accuracy of the translation operator and is selected empirically, often using formulas like the Excess Bandwidth Formula (EBF) [42]. Although EBF is beyond the scope of this thesis, its output value is used.

MLFMA performs MVM in three main stages:

- **Aggregation:** Outgoing fields from basis functions are projected onto spherical harmonics and recursively collected from the leaf level up to the root. The truncation number τ varies with box size, requiring interpolation when moving between levels.
- **Translation:** Interactions between well-separated groups at the same level are computed using translation operators. The number of spherical modes and angular sampling points, governed by τ, dictates the accuracy of this stage.

- **Disaggregation:** Incoming fields are distributed from root to leaves, where they are evaluated on the testing functions. Anterpolation ensures accurate mapping across resolution changes.

By decoupling geometric and translation-dependent computations, and recursively applying these stages at multiple levels, MLFMA achieves high efficiency in simulating large-scale electromagnetic problems while maintaining accuracy.

2.2 Coding-Related Preliminaries

While electromagnetic solvers have traditionally been developed in environments like MATLAB or Fortran, recent advances in Python's scientific computing ecosystem have made it a viable alternative for implementing high-performance numerical algorithms. MATLAB is widely used for electromagnetic simulation prototyping due to its built-in vectorization and visualization capabilities. Contrary to some misconceptions, MATLAB inherently includes a just-in-time (JIT) compilation engine, which enables runtime optimization of script execution [43]. As a result, integrating JIT compilation into Python using tools like `Numba` does not intrinsically offer a performance edge over MATLAB's native execution. However, MATLAB also supports MEX (MATLAB Executable) files, which allow critical parts of the code to be written in low-level languages such as C or C++ and compiled for significantly improved performance. When leveraged properly, MEX files can outperform both standard MATLAB scripts and Python implementations, especially in compute-intensive loops or memory-bound operations [44]. Nonetheless, these capabilities often come with added complexity in setup and platform dependency, and advanced parallelization features in MATLAB typically require additional toolboxes [45].

In the first part of this thesis study, Python is selected for MLFMA simulations due to its flexibility, open-source nature, and growing support for high-efficiency computing. Its ecosystem of high-performance libraries, such as `NumPy`, `Numba`, and `CuPy`, enables JIT compilation, CPU parallelization, and GPU acceleration.

These capabilities are critical for reducing the runtime of computationally expensive routines in the MLFMA framework. The following subsections outline the optimization techniques employed to enhance the performance of our Python-based implementation.

2.2.1 Just-In-Time (JIT) Compilation

JIT compilation is a dynamic optimization technique that translates high-level programming code into machine code at runtime, significantly reducing execution time for numerically intensive computations. In Python-based scientific computing, the `Numba` library offers an Low Level Virtual Machine (LLVM) based JIT compiler that enables Python functions to be accelerated via a single `@jit` decorator, which can result in performance close to compiled C or Fortran codes [36]. In the context of our implementation, JIT compilation is employed across all stages of MLFMA to minimize the overhead of repeated mathematical operations, particularly in recursive computations involving translation operators and spherical harmonics.

2.2.2 CPU Parallelization

Modern CPUs contain multiple cores, allowing for task-based or data-parallel operations to be distributed across threads or processes. In Python, this capability is effectively leveraged using parallel backends such as `Numba`'s `prange` directive or the `multiprocessing` module [31, 36]. In this thesis, we implement CPU parallelization specifically in the near-field computation stage of MLFMA, which is typically the most time-consuming part due to the large number of pairwise interactions. By distributing the computation of near-field matrix entries across multiple threads, significant reductions in execution time are achieved without compromising accuracy.

2.2.3 GPU Parallelization

GPUs are designed for massive data-parallel operations and are especially suited for algorithms involving large-scale linear algebra and vectorized data structures. CUDA-enabled GPUs allow developers to offload computational kernels directly to the device, bypassing CPU limitations [28,35]. Although GPU acceleration has not been fully integrated into our current MLFMA implementation, its suitability, particularly for the translation and aggregation stages, is promising. Future work may involve leveraging `Numba`'s `cuda.jit` interface or frameworks like `CuPy` to achieve fine-grained GPU acceleration of matrix-vector products [27, 37].

Chapter 3

Performance Optimization of MLFMA

3.1 Just-In-Time (JIT) Compilation

JIT compilation is a runtime optimization technique designed to bridge the gap between the development flexibility of high-level interpreted languages and the performance of low-level compiled code. Unlike traditional ahead-of-time (AOT) compilation, which translates the entire source code into machine code before execution, JIT compilation delays this process until the code is actually executed. This allows the compiler to optimize the generated machine code based on runtime information, such as input data types, memory access patterns, and control flow behavior [46].

In Python, one of the most widely used JIT tools is the `Numba` library, which leverages the LLVM compiler infrastructure to transform Python functions into highly optimized machine code [36]. This transformation occurs through the use of decorators such as `@jit` or `@njit`, which compile decorated functions the first time they are called. From that point on, all future invocations of the same function operate using the compiled machine-level code, significantly accelerating

the runtime performance.

One of the key advantages of JIT compilation in scientific computing is its ability to reduce the overhead associated with Python's interpreted execution model. For numerical operations that involve repeated loops, array manipulations, and element-wise computations, the overhead of Python's dynamic type checking and bytecode interpretation can become substantial. JIT compilation bypasses this bottleneck by producing native CPU instructions that eliminate the need for runtime interpretation, thus resulting in performance that is often comparable to statically compiled languages such as C or Fortran.

In the context of our MLFMA implementation, JIT compilation was selectively applied to the most time-consuming and computation-heavy routines. These include:

- **Hierarchical Tree Construction:** The recursive generation of the MLFMA's multilevel spatial decomposition, which involves iterating over elements and assigning them to hierarchical subdomains.
- **Near-Field Interaction Setup:** The definition and storage of spatially close interactions, which require dense loop structures and geometric distance checks.
- **Full MLFMA Solver:** The complete execution of MLFMA including aggregation, translation, and disaggregation operations, composed of nested iterations and mathematically intensive steps.
- **Iterative Generalized Minimal Residual (GMRES) Solver:** The repeated application of matrix-vector products in the GMRES algorithm, used to approximate the final solution iteratively.

These operations are characterized by recursive logic, intensive loop structures, and large-scale numerical computations, making them ideal candidates for JIT acceleration. By compiling these routines with `Numba`, we were able to significantly reduce runtime while preserving the flexibility of Python's high-level syntax.

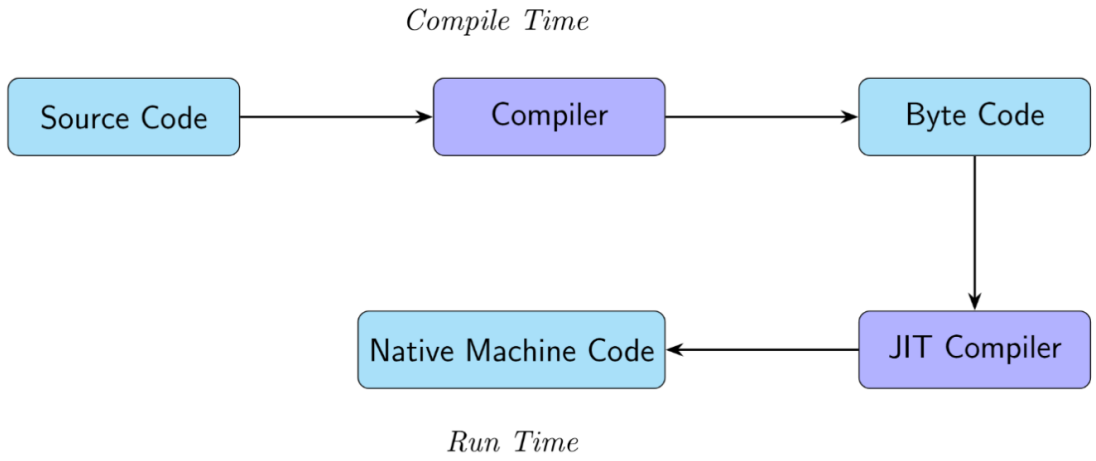


Figure 3.1: Workflow of JIT compilation and its integration into the runtime execution pipeline.

The workflow of JIT compilation and its integration into the execution pipeline is illustrated in Figure 3.1. This diagram highlights the transformation from high-level source code to native machine code, emphasizing the deferred nature of compilation and the runtime optimization benefits it provides.

It is also worth noting that JIT compilation integrates well with NumPy-based data structures, which are already optimized for efficient numerical storage and vectorized operations. `Numba` automatically recognizes many NumPy array operations and translates them into equivalent LLVM-based loops, further minimizing the overhead and maximizing data throughput [31]. Additionally, the use of the `nopython=True` option ensures that all operations within the decorated function are compiled, and no Python interpreter fallback is allowed, this typically yields the highest performance gains.

Overall, JIT compilation offers a low-effort yet highly effective means of accelerating numerical Python code. In our implementation of MLFMA, it enabled near-C-level speedup in multiple computation stages, laying a strong foundation for further parallelization efforts via CPU multithreading and GPU acceleration.

3.2 Numba Parallelization (CPU)

Parallel computing has become an essential paradigm for addressing the computational demands of large-scale scientific simulations. Traditional serial processing models are inherently limited by the sequential nature of their execution, which often becomes a bottleneck when dealing with high-dimensional numerical problems such as those encountered in computational electromagnetics. In contrast, parallel computing allows tasks to be distributed across multiple processing cores or threads, enabling simultaneous execution of independent operations and thereby significantly reducing overall runtime [45].

Within the context of Python-based scientific computing, the `Numba` library provides a lightweight yet powerful approach to implementing parallel execution on multicore CPUs. In addition to its JIT compilation capabilities, `Numba` offers parallelization support through the use of the `parallel=True` keyword and parallel-safe NumPy constructs such as `prange` [31, 36]. By compiling decorated functions into multithreaded native code, `Numba` can leverage multiple CPU cores to accelerate numerically intensive tasks without requiring explicit thread management by the programmer.

In our implementation of MLFMA, the computational workload is naturally divided into two distinct regions: near-field and far-field interactions. The near-field region comprises interactions between geometrically adjacent basis functions, and it requires the direct evaluation of dense interaction matrices. These evaluations involve pairwise computations of Green's function interactions over thousands (or even millions) of basis function pairs and typically dominate the overall computation time in MLFMA-based simulations [18]. While the far-field computations benefit from the hierarchical decomposition and translation mechanisms intrinsic to MLFMA, they are significantly less expensive in terms of raw floating-point operations per element. On the other hand, near-field computations are local and inherently parallelizable, as the calculations for each near-box interaction are independent of one another. Therefore, in addition to employing JIT compilation for low-level arithmetic optimizations, we also incorporated CPU parallelization

via `Numba` for accelerating this segment of the algorithm. Specifically, we identified the routines responsible for near-field matrix construction and modified their iterative loops to use `prange` instead of standard Python `for` loops. This change enabled automatic thread-level parallelism under the `Numba` backend, which dynamically scheduled the workload across available CPU cores. In conjunction with the `nopython=True` and `fastmath=True` options, we ensured that the compiled routines executed with minimal overhead and maximized throughput.

The effectiveness of this parallelization strategy was evident in our benchmarks, where the runtime associated with near-field matrix evaluation was reduced substantially. As a result, the total simulation time for large-scale electromagnetic problems decreased considerably, especially in scenarios where the geometry involved a high density of RWG basis functions and thus an extensive set of near-field interactions.

Ultimately, by combining JIT compilation with CPU-based parallelism, we achieved substantial performance improvements without compromising the simplicity and flexibility of Python. This hybrid approach allowed us to exploit both algorithmic and architectural parallelism, paving the way for further extensions using GPU acceleration and distributed-memory techniques in future implementations.

3.3 Time Comparisons

Performance benchmarking is a crucial step in evaluating the effectiveness of algorithmic optimizations, especially in high-performance computing applications such as MLFMA. In this section, we present a series of timing comparisons to quantify the computational benefits of the enhancements applied in our implementation. These enhancements primarily include JIT compilation and CPU-based parallelization using the `Numba` library.

The goal of these benchmarks is to assess the real-world impact of the proposed optimizations on overall runtime. We compare execution times across different programming environments and optimization configurations, including native MATLAB implementations, unoptimized Python baselines, and optimized Python versions with JIT and parallel execution. The test scenarios are chosen to reflect typical use cases in electromagnetic simulations, focusing on representative problem sizes and operational conditions.

3.3.1 JIT Time Results

3.3.1.1 Constructing the Hierarchical Tree

One of the core components of MLFMA is the construction of its hierarchical tree structure. This structure is used to organize the spatial domain into multiple levels of nested cubes, allowing for efficient aggregation and translation operations. The corresponding implementation function responsible for this process is named `constructtree()`.

Although the generation of the tree structure is straightforward for simple and uniformly meshed objects, the construction process becomes computationally intensive for geometries that are large, complex, or irregularly shaped. The increasing number of levels and non-empty boxes in the tree significantly amplifies the processing time, particularly in Python due to its interpreted execution model. To demonstrate how tree construction affects processing time, we consider a relatively simple geometry: a sphere with a diameter of 0.6 meters, illuminated at an operational frequency of 251 MHz, which corresponds to a wavelength of approximately 1.2 meters. Hence, the electrical size of the object is $\lambda/2$. The sphere is discretized using 4018 triangular elements, resulting in 6027 RWG basis functions with an average edge length of 25 mm ($\lambda/48$). For numerical integration, each basis function is integrated using three-point quadrature, while a single integration point is employed for the testing functions. Fig. 3.2 shows the triangular mesh representation of the object.

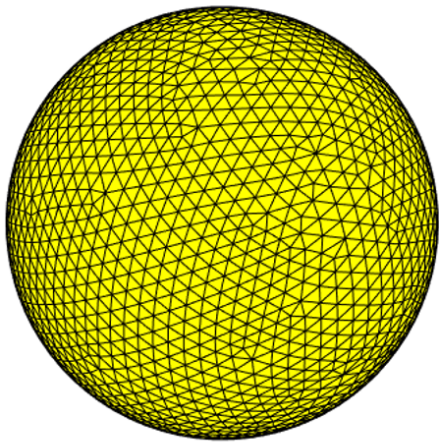


Figure 3.2: Mesh representation of the test sphere using uniform triangulation. The geometry is discretized with RWG basis functions having an average edge length of 25 mm ($\lambda/48$).

All timing results reported in this section are obtained on the same machine equipped with an Intel Core i9-13950HX processor, an NVIDIA RTX 4080 Laptop GPU, and 32 GB of DDR5 RAM. The execution times for constructing the hierarchical tree in different environments are summarized in Table 3.1.

Table 3.1: `constructtree()` timing results for the sphere geometry across different platforms.

Implementation	**Real Time (sec)**
Python (Original)	1.61
Python (JIT)	0.043
MATLAB (no MEX)	0.02

As shown in Table 3.1, the original Python implementation exhibits a relatively high runtime compared to the MATLAB baseline. However, the application of JIT compilation dramatically reduces this cost, producing a runtime close to that of MATLAB. Although the time savings in this case may appear moderate due to the simplicity of the geometry, they establish a promising baseline for

more complex structures. The MEX version is not included in this table because `constructtree()` defines many fundamental variables whose sizes are dynamically determined at runtime. Since MEX-compatible code generation requires all variable sizes to be known and fixed at compile time, a MEX version of this function could not be created.

To demonstrate the impact of JIT compilation in a more demanding scenario, we next examine the case of a Flamme geometry—a sharp, elongated object with intricate surface features. The object has an overall length of approximately 0.6 meters and is illuminated at an operational frequency of 251 MHz, which corresponds to a wavelength of approximately 1.2 meters. Hence, the electrical size of the object is $\lambda/2$. It is discretized using 34,446 triangular elements, resulting in a total of 51,669 RWG basis functions. Each basis function is integrated using three-point quadrature, while a single integration point is used for testing functions. As in the previous example, all simulations are performed on the aforementioned same machine. The corresponding meshed flamme geometry is shown in Figure 3.3.

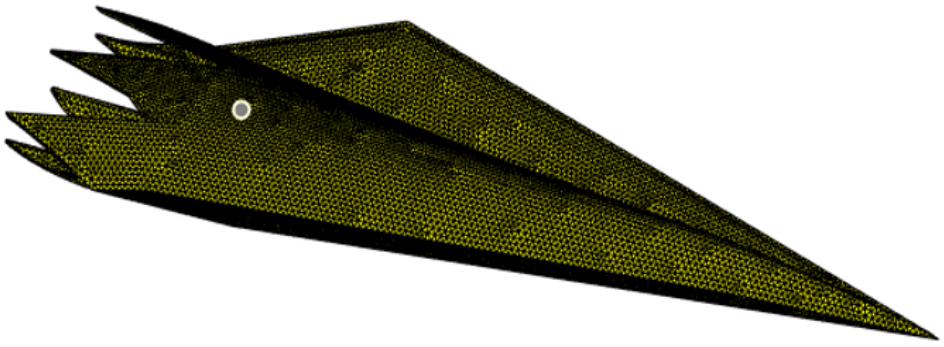


Figure 3.3: Mesh representation of the Flamme geometry

The tree construction runtimes for this geometry are given in Table 3.2.

Table 3.2: Timing results of the `constructtree()` function for the Flamme geometry across different platforms.

Implementation	**Real Time (sec)**
Python (Original)	199.2028
Python (JIT)	4.3971
MATLAB (no MEX)	0.13

The performance gain in this case is relatively better. The JIT-compiled version reduces the runtime by more than a factor of 45 compared to original Python, highlighting the significant performance gain of JIT when dealing with large and complex geometries. Notably, the JIT-accelerated implementation also approaches the performance of a highly optimized MATLAB implementation. Similar to the sphere case, the MEX version is not included in this table because `constructtree()` defines many fundamental variables whose sizes are dynamically determined at runtime. Since MEX-compatible code generation requires all variable sizes to be known and fixed at compile time, a MEX version of this function could not be created.

These results clearly demonstrate the effectiveness of JIT compilation in reducing the computational cost of hierarchical tree construction in MLFMA. When applied to real-world electromagnetic problems involving complex geometries, JIT compilation transforms Python from a prototyping tool into a viable high-performance simulation environment.

Despite the considerable speedup achieved through JIT compilation, MATLAB remains significantly faster than the Python-based implementations, often by a factor of 30 or more. This performance gap can be explained by several key factors. First, MATLAB leverages highly optimized, compiled low-level libraries (e.g., BLAS/LAPACK and other routines written in C or Fortran) for core array operations, offering superior performance even on interpreted code paths [47]. In contrast, Python's NumPy and Numba combination still incurs interpreter overhead for memory management, function calling, and runtime dispatch [48].

Second, MATLAB's array engine is designed for optimal memory locality and efficient use of Single Instruction, Multiple Data (SIMD) vector instructions [49,50], while Python/NumPy often requires explicit programming patterns to approach this level of hardware utilization [48]. Third, while MATLAB applies JIT optimizations broadly across built-in functions, Python's Numba must be applied manually and only accelerates the user-decorated segments of code, leaving other parts (e.g., array setup and I/O) to run in slower interpreted mode [48]. These architectural differences, tightly integrated native libraries, automated optimization, and comprehensive JIT, explain why MATLAB continues to outperform Python in these benchmarks.

3.3.1.2 Constructing Near-Field Interactions

Another critical stage in the MLFMA pipeline is the construction of near-field interaction lists between RWG basis functions. While MLFMA significantly reduces computational complexity by approximating far-field interactions via groupwise operations, it still requires the explicit evaluation of all pairwise interactions within the near-field region. These interactions are computed based on a one-box-buffer scheme, which considers neighboring boxes to identify near-field RWG pairs.

For dense meshes or large geometries, the number of such near-field interactions can still reach thousands or even millions, and every pair must be identified and evaluated. The corresponding routine, `constructnearfield()`, involves several nested loops and condition checks to enumerate all valid near-box RWG combinations. As a result, it becomes the most time-consuming stage in our MLFMA implementation when executed without optimization. To mitigate this bottleneck, we applied JIT compilation to this routine as well. Due to its highly repetitive and arithmetic-heavy structure, the function was an ideal candidate for compilation. In this case, we focus solely on the aforementioned sphere example with 25 mm RWG mesh, as it already contains a sufficiently large number of RWG pairs to highlight the performance differences across implementations.

Table 3.3: `constructnearfield()` timing results for the sphere geometry

Implementation	**Real Time (sec)**
Python (Original)	10698.47 (2.97 hours)
Python (JIT)	222.99
MATLAB (no MEX)	185.46
MATLAB (MEX-accelerated)	6.36

As seen in Table 3.3, the original Python implementation required nearly three hours to complete this stage. With the addition of JIT compilation, the runtime was drastically reduced to under four minutes, offering more than 45 times speedup. Although this improvement brings Python performance close to that of MATLAB, the latter still maintains a slight edge, likely due to its lower-level memory management and mature built-in vectorization for matrix assembly. Nevertheless, the results confirm that JIT compilation transforms an otherwise impractical implementation into a viable one for real-world MLFMA applications. In particular, optimizing the near-field construction routine significantly improves the overall feasibility of large-scale simulations in Python, especially when combined with other acceleration techniques.

3.3.1.3 MLFMA Solver and Generalized Minimal Residual (GMRES) Iteration Loop

The final stage of our MLFMA pipeline involves solving the system of linear equations $\bar{\bar{Z}}\bar{\alpha} = \bar{v}$. Due to the large-scale nature of electromagnetic scattering problems, direct solvers are impractical in terms of both time and memory. Instead, iterative solvers such as GMRES are commonly employed to find approximate solutions [51]. The GMRES algorithm operates by successively applying the MVM $\bar{\bar{Z}}\bar{\alpha}$ through a user-defined function handle, rather than storing $\bar{\bar{Z}}$ explicitly. In our case, this function handle corresponds to the core `mlfma()` routine, which encapsulates the full multilevel aggregation, translation, and disaggregation operations required to compute the effect of $\bar{\bar{Z}}$ on a given vector $\bar{\alpha}$. As GMRES

iterates, it repeatedly invokes this function, making the efficiency of `mlfma()` critically important. Given that both `mlfma()` and the surrounding GMRES loop involve recursive mathematical operations and nested loops, they were both compiled using Numba's JIT engine to improve performance. The benefits of this optimization become evident in timing results obtained from the aforementioned sphere geometry with approximately 25 mm RWG edge lengths.

Table 3.4: Timing results for GMRES solver using `mlfma()` as matrix-vector operator (38 iterations)

Implementation	**Number of Iterations**	**Real Time (sec)**
Python (Original)	38	4559.098 (1.27 hours)
Python (JIT)	38	7.87
MATLAB (no MEX)	38	26.10
MATLAB (MEX-accelerated)	38	8.4

Table 3.4 demonstrates that the original Python solver took over three hours to complete 38 GMRES iterations. After incorporating JIT compilation, the runtime was reduced to under 8 seconds, significantly outperforming even the MEX-accelerated MATLAB implementation.

These results highlight the compounded advantage of JIT compilation when applied to both the iterative solver and its underlying matrix-vector product operator. By optimizing the GMRES loop and the `mlfma()` routine together, we achieved substantial reductions in execution time while maintaining numerical accuracy, paving the way for interactive or real-time simulations in moderate-scale configurations.

3.3.2 CPU Parallelization Time Results

As presented in the timing results of the `constructnearfield()` routine in the previous subsection, the evaluation of near-field interactions remains one of the most time-consuming components in the MLFMA framework, even after applying

JIT compilation. This computational burden arises from the need to exhaustively compute pairwise interactions between a large number of adjacent RWG basis functions, leading to deeply nested loop structures that dominate execution time. To further mitigate this bottleneck, CPU-based parallelization using `Numba`'s multithreading capabilities was employed. Specifically, the iterative loops within the near-field construction routine were parallelized using `prange`, allowing the workload to be distributed across multiple CPU cores. This approach exploits the embarrassingly parallel nature of the problem, as each pairwise RWG interaction can be computed independently.

To evaluate the performance improvements gained through CPU parallelization, we conducted a series of experiments using varying numbers of CPU cores. The resulting runtimes are summarized in Table 3.5, and the corresponding performance trend is illustrated in Figure 3.4.

Table 3.5: Runtime of `constructnearfield()` with different numbers of CPU cores

Number of Cores	**Runtime (sec)**
5	47.737
10	27.576
15	17.802
20	14.510
25	13.680
30	12.235
32	11.516

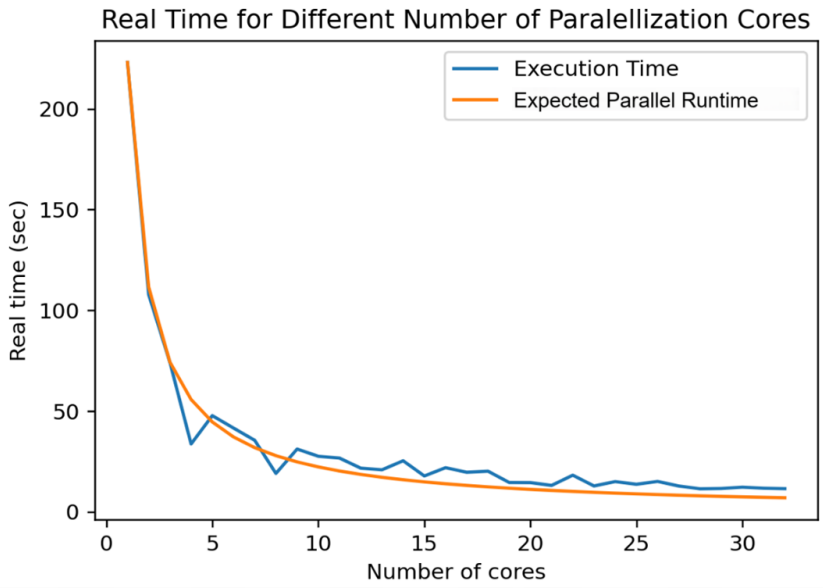


Figure 3.4: Performance comparison between actual execution time and expected parallel runtime based of number cores.

The plot in Fig. 3.4 includes two curves: the actual measured execution time and the so-called *expected parallel runtime*. It is important to clarify that the latter is not a theoretical limit derived from an analytical model, but rather a baseline estimate computed by dividing the single-core (JIT-only) execution time by the number of cores used. This simplification assumes perfect linear scaling and no parallelization overhead, which is rarely achievable in practice due to thread synchronization, memory bandwidth contention, and task scheduling inefficiencies. Nevertheless, the empirical results closely follow the idealized curve, particularly for smaller core counts, indicating that the parallelized implementation scales effectively. Beyond a certain threshold, the performance gain begins to saturate, suggesting the onset of diminishing returns due to hardware-level limitations.

It is also worth noting that the same `constructnearfield()` routine was implemented in MATLAB at Section 3.3.1.2. Although native parallelization via `parfor` could not be effectively applied due to complex indexing patterns in the nested loops, significant performance gains were still achieved through

MEX acceleration. Specifically, the runtime was reduced from approximately 186 seconds in the original MATLAB implementation to around 7 seconds with MEX support, as shown in Table 3.3. In comparison, the Python implementation with both JIT compilation and 32-core CPU parallelization achieved a runtime of approximately 11 seconds. This is slightly slower than the MEX-accelerated MATLAB version, but still very close. In addition, the Python version is easier to modify, works on different systems without change, and can handle larger problem sizes more easily thanks to its flexible parallel programming support. With further optimization and a more in-depth parallelization strategy, it may even be possible to surpass the performance of the current MEX-accelerated MATLAB version.

In summary, parallelizing the `constructnearfield()` routine using CPU multithreading produced a significant improvement in performance over the already optimized JIT version. This further validates the modular nature of the MLFMA pipeline and the benefits of combining algorithmic parallelism with hardware-aware execution strategies.

3.4 Future Directions

MATLAB is widely used for rapid prototyping and offers built-in support for GPU acceleration through its Parallel Computing Toolbox [52]. However, despite these advantages, it presents several limitations in the context of large-scale, research-oriented electromagnetic solvers. First, MATLAB's GPU support is restricted to high-level operations and built-in functions, which limits low-level memory control and thread-level optimization on the GPU [52]. In contrast, Python, via libraries such as `Numba` [53], `CuPy`, and `PyCUDA` [54], allows direct access to GPU kernels, shared memory, and thread/block configuration, which are essential for implementing custom electromagnetic algorithms like MLFMA. Second, MATLAB is a closed-source, commercial platform with licensing constraints, especially regarding GPU toolboxes and deployment at scale. Python, being open-source and free, offers broader accessibility and ease of integration into High-Performance Computing (HPC) environments [55] and research pipelines. Finally, transitioning to

Python aligns with modern trends in scientific computing, where deep learning frameworks, GPU programming, and hardware-aware compilation workflows increasingly rely on Python-based ecosystems. Therefore, this work adopts Python to enable deeper customization, fine-grained performance tuning, and long-term extensibility beyond what MATLAB currently affords.

While the current implementation leverages `Numba`'s CPU-based parallelization to accelerate computationally intensive routines, further performance gains can potentially be achieved through the use of GPU acceleration. `Numba` offers a separate CUDA-targeted module that allows Python code to be compiled and executed directly on NVIDIA GPUs. This module provides support for kernel programming, shared memory usage, and efficient thread-level parallelism, making it well-suited for highly repetitive, data-parallel tasks such as those found in electromagnetic solvers [53].

Compared to CPUs, GPUs offer a significantly larger number of cores, often thousands per device, enabling massive parallelism for workloads that can be distributed across many threads [28]. In the context of the MLFMA framework, routines such as near-field matrix construction, translation operator evaluations, and aggregation/disaggregation steps are inherently parallelizable and could benefit from GPU acceleration. Specifically, these tasks involve large volumes of independent floating-point operations and exhibit minimal data dependencies, which align well with the execution model of GPUs.

Additionally, iterative solvers like GMRES, which require repeated application of matrix-vector operations, may also see improved throughput on GPUs, especially when the associated operators (e.g., the `mlfma()` routine) are offloaded and optimized for execution on CUDA cores.

However, integrating GPU support requires careful consideration of memory management and data transfer overhead between the host (CPU) and the device (GPU). The benefits of GPU acceleration become more pronounced for larger problems where the cost of data transfer is amortized over significant computation. Moreover, memory access patterns must be optimized to take full advantage

of GPU architecture, including the use of coalesced accesses and shared memory.

As a future direction, exploring GPU acceleration for the MLFMA pipeline using `Numba`'s CUDA functionality may be considered. This includes identifying computational hotspots that can be offloaded, redesigning data structures for device compatibility, and benchmarking performance against existing CPU-parallel implementations. Such an extension holds promise for enabling real-time simulation capabilities and extending the applicability of our method to extremely large-scale electromagnetic problems.

Chapter 4

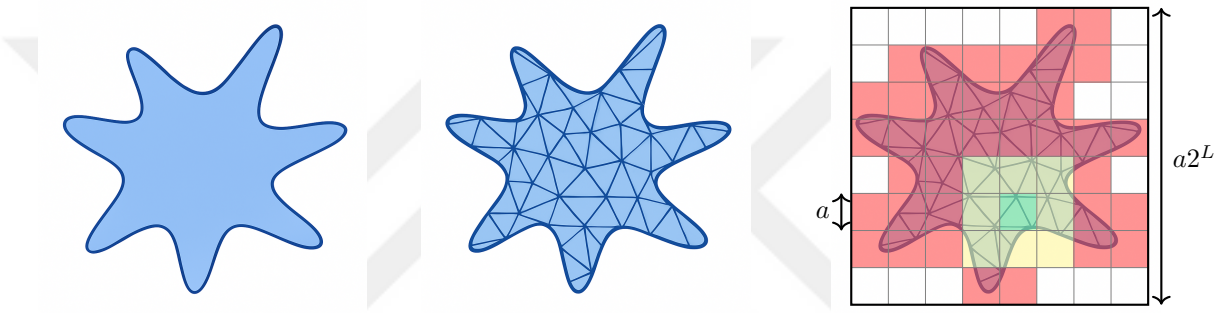
Surface Dipole Method

4.1 Formulations

As discussed in Chapter 2, discretization of the integral equation using MoM leads to the matrix system shown in equation (2.41). To alleviate the computational cost associated with this system, we adopt a strategy inspired by MLFMA, where the total interaction matrix is conceptually divided into near-field and far-field components, as expressed in equation (2.47). While this decomposition resembles the structure employed in MLFMA, the way far-field interactions are computed in our method follows a fundamentally different approach, as described in [29] and below.

The method begins by enclosing the entire computational domain within a cubic volume referred to as the *mother box*, which is uniformly subdivided into 8^L smaller cubes, called *child boxes*, at a user-defined decomposition level L. Each child box has an edge length of a, so that the mother box spans a total edge length of $a2^L$. The decomposition level is selected to satisfy the condition $a2^{L-1} < D \leq a2^L$, where D denotes the diameter of the object, ensuring full coverage of the geometry.

A one-box-buffer scheme is employed to classify interactions between boxes as either near-field (NF) or far-field (FF). Specifically, interactions involving a given source box and its immediate neighbors (including diagonal neighbors) are treated as near-field and computed directly. All remaining interactions are considered far-field and are evaluated using the proposed acceleration strategy. Figure 4.2 illustrates this NF/FF separation.



(a) (b) (c)

Figure 4.1: Illustration of the object: (a) original geometry, (b) RWG discretization, and (c) Spatial decomposition of the geometry into an 8×8 grid. The central green cell denotes the selected source box. Yellow cells represent its near-field (NF) neighbors, while red cells correspond to the far-field (FF) region. Unshaded cells indicate empty boxes with no interaction.

Rather than forming the far-field matrix explicitly, we compute its effect on the unknown vector through a modular three-stage procedure [6, 29]:

1. **Mapping to Uniform Basis Functions:** Subdomain basis functions (RWGs) located in each non-empty box are mapped to a fixed number of arbitrarily oriented and uniformly distributed basis functions, hereafter referred to as uniform basis functions.

2. **Translation:** Electromagnetic interactions between dipole groups are computed using the free-space Green's function evaluated at relative translation vectors, enabling efficient far-field interaction calculations.

3. **Inverse Mapping to Subdomain Functions:** The fields computed at test locations are mapped back to the subdomain test functions within each box in a geometry-independent manner.

This formulation enables efficient evaluation of far-field interactions across large problem sizes, while preserving the accuracy required by the underlying physical model. Each stage is described in more detail in the following subsections.

4.1.1 Mapping to Uniform Basis Functions

To initiate the construction of a uniform basis model, the scatterer is first divided into small cubic boxes, each containing a number of RWG basis functions depending on the local surface geometry. When evaluating the electromagnetic field produced by these basis functions at a distance that qualifies as far field relative to the box's size (e.g., outside a one-box buffer zone), it is essential to determine the field strength in various directions. This is accomplished by computing the electric field at a set of observation points located on a virtual sphere surrounding the box, providing directional sampling of the radiated field.

In the first mapping stage, the goal is to determine the current coefficients of a fixed number of arbitrarily oriented and uniformly distributed Hertzian dipole basis functions within each box. These dipole basis functions are used to represent the subdomain RWG basis functions in a structured and geometry-independent manner. To achieve this, the far-field behavior of the RWG basis functions is sampled at a set of observation points surrounding each box, and the current coefficients of the dipole basis functions are computed such that the resulting dipole-generated fields match those of the original RWG functions at these far-field sampling locations.

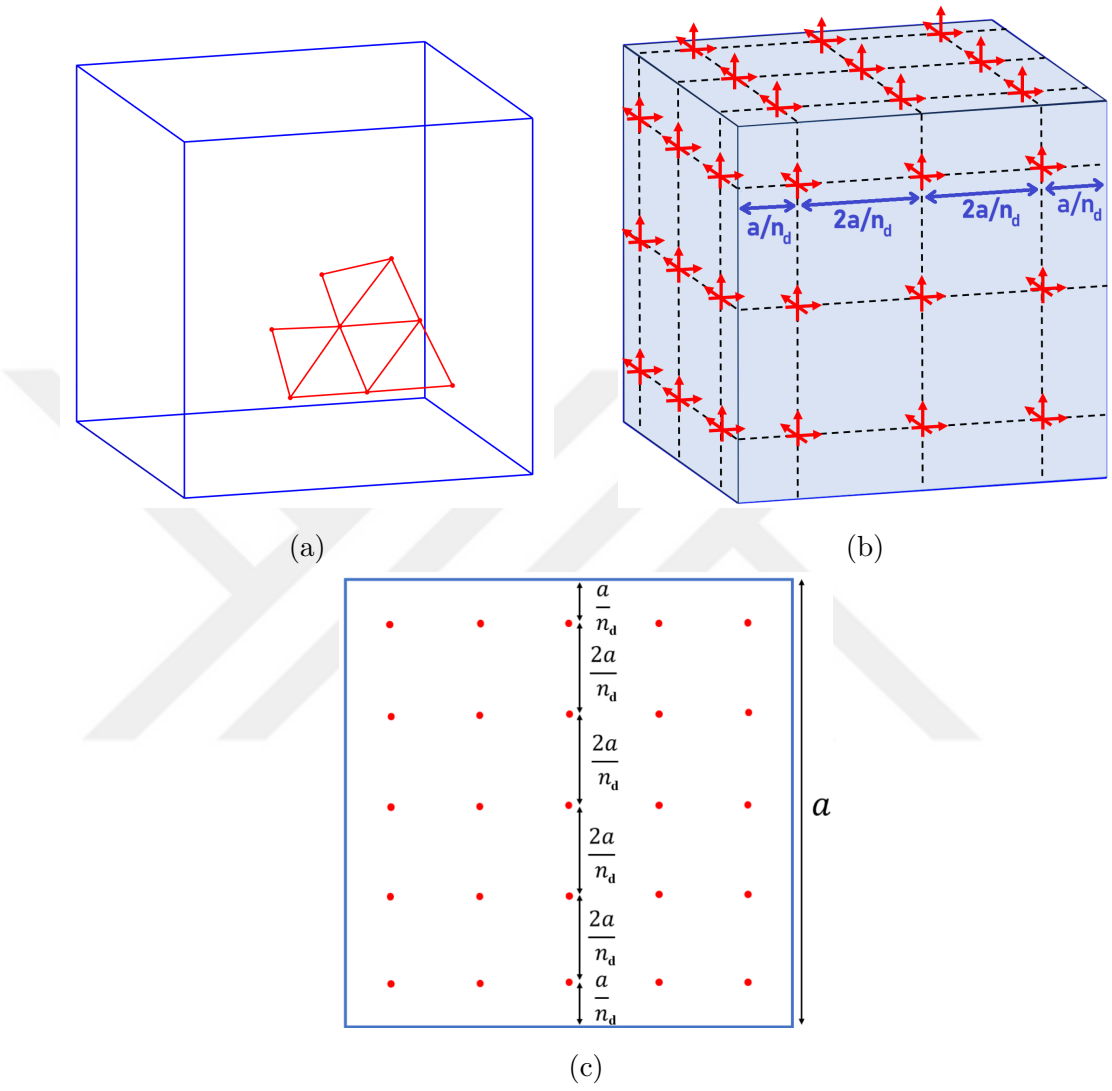


Figure 4.2: (a) A group of RWG basis functions randomly located within a box, (b) uniformly distributed surface dipoles used as an alternative basis.

A related formulation was recently proposed in [29], where the original RWG basis functions within each box are represented using a fixed number of Hertzian dipoles that are arbitrarily oriented and uniformly distributed throughout the volume. This volumetric approach removes mesh dependency, provides a consistent local basis for all boxes, and enables efficient parallelization by enforcing identical dipole configurations across subdomains.

In the method represented in this thesis, to enable a structured and consistent representation across all boxes, a surface-based strategy is employed in which Hertzian dipoles are uniformly distributed at predefined positions on the surfaces of each non-empty box. This configuration ensures that each box uses an identical set of basis functions, independent of the underlying mesh, thereby eliminating the variability introduced by irregular RWG basis function distributions, similar to the objective of the volumetric approach in [29]. As a result of this design choice, the following outcomes are expected

- Achieving slightly more accuracy comparable to that of the volumetric method.
- Gaining a significant dipole count advantage as the box size increases.

The equivalent current distribution confined within a child box can be described as a weighted sum of the local basis functions defined over that subdomain. If $\bar{b}_n(\bar{r})$ denotes the n-th RWG basis function and $\alpha_n = a[n]$ its corresponding coefficient, the total current in the box is given by [29]

$$\bar{J}(\bar{r}) = \sum_{n=1}^{N_b} \alpha_n \bar{b}_n(\bar{r}), \tag{4.1}$$

where N_b represents the number of RWG basis functions inside the selected box. These coefficients α_n are subject to iterative updates during the solution process. The resulting electric field generated by this localized current distribution in (4.1) at an observation point $\bar{r}$ is calculated by considering all source points $\bar{r}'$ within the box [29] given by

$$\bar{E}_0(\bar{r}) = \int \bar{\bar{D}}(\bar{r}, \bar{r}') \cdot \bar{J}(\bar{r}')\, d\bar{r}'. \tag{4.2}$$

In (4.2), $\bar{\bar{D}}(\bar{r}, \bar{r}')$ is the free-space dyadic Green's function that governs the electromagnetic interaction between the source and observation points in homogeneous space. It is expressed as

$$\bar{\bar{D}}(\bar{r}, \bar{r}') = -jk\eta \frac{e^{-jkR}}{4\pi R} \left[\bar{\bar{I}} \left(1 - \frac{j}{kR} - \frac{1}{k^2R^2} \right) - \hat{R}\hat{R} \left(\frac{3j}{kR} + \frac{3}{k^2R^2} \right) \right], \tag{4.3}$$

where $\bar{R} = \bar{r} - \bar{r}'$, $R = \|\bar{R}\|$, and $\hat{R} = \bar{R}/R$ is the unit vector pointing from the source to the observation point. The parameters k and η correspond to the free-space wavenumber and intrinsic impedance, respectively.

As illustrated in Fig. 4.2, a corresponding representation is defined for the uniform basis functions, which consist of a fixed number of Hertzian dipoles placed at predetermined positions and aligned along the Cartesian directions $\hat{x}$, $\hat{y}$, and $\hat{z}$. The equivalent current distribution of these basis functions is then formulated as [29]

$$\bar{J}_d(\bar{r}) = \sum_{\hat{a}\in\{\hat{x},\hat{y},\hat{z}\}} \sum_{k=1}^{N_d} \beta_{\hat{a},k}\, \bar{p}_{\hat{a},k}(\bar{r}) \tag{4.4}$$

where $\bar{p}_{\hat{a},k}(\bar{r})$ denotes the k-th dipole basis function oriented along direction $\hat{a}$ and placed at a fixed position, and $\beta_{\hat{a},k}$ is its unknown coefficient to be determined during the mapping process. The vectorial orientation of these dipole functions is explicitly defined as

$$P_{\hat{x},k} = \begin{bmatrix} p_k \\ 0 \\ 0 \end{bmatrix}, \quad P_{\hat{y},k} = \begin{bmatrix} 0 \\ p_k \\ 0 \end{bmatrix}, \quad P_{\hat{z},k} = \begin{bmatrix} 0 \\ 0 \\ p_k \end{bmatrix}.$$

The total number of dipoles N_d is kept constant across all boxes to ensure structural uniformity, which leads to computational efficiency for parallelization. Each dipole has three vectorial components, resulting in $3N_d$ unknown coefficients in total.

The electric field radiated by this uniform current distribution at a given observation point $\bar{r}$ is obtained by superposing the contributions from all dipole basis functions via the free-space dyadic Green's function given by [29]

$$\bar{E}_d(\bar{r}) = \sum_{\hat{a}\in\{\hat{x},\hat{y},\hat{z}\}} \sum_{k=1}^{N_d} \beta_{\hat{a},k} \int \bar{\bar{D}}(\bar{r},\bar{r}') \cdot \bar{p}_{\hat{a},k}(\bar{r}')\, d\bar{r}'. \tag{4.5}$$

According to the electromagnetic uniqueness theorem, if two different source distributions produce identical electric fields on a closed surface, then the fields they produce outside that surface must also be identical. Leveraging this principle, the proposed method seeks to determine the current coefficients $\beta_{\hat{a},k}$ of the

dipole-based uniform basis functions by minimizing the difference between the electric field generated by the original RWG functions and that of the dipole configuration over a spherical observation surface surrounding each box.

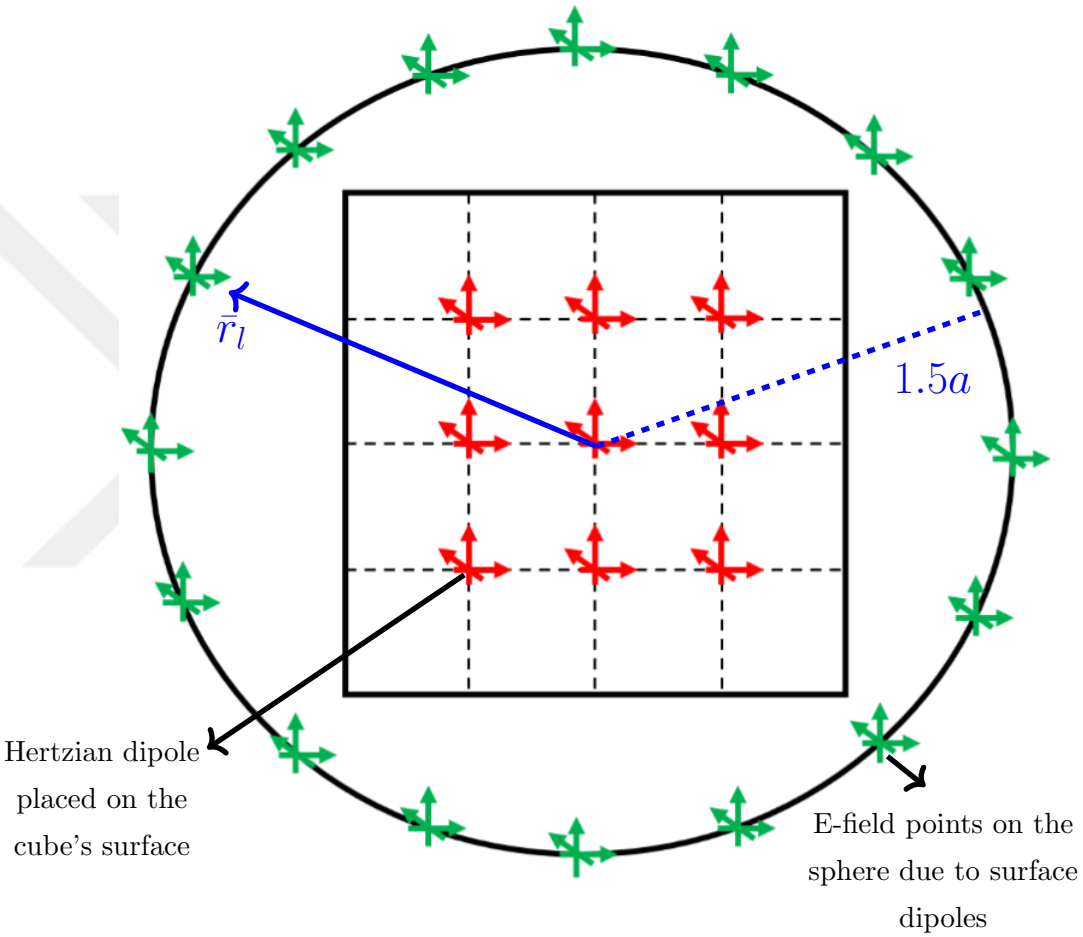


Figure 4.3: Red arrows represent Hertzian dipoles uniformly distributed on the visible surface of the cube, while green arrows denote the electric field samples computed on a surrounding sphere. The setup illustrates the far-field evaluation due to surface-based dipole sources.

To ensure that the matching surface fully captures the far-field behavior of the RWG basis functions without violating the one-box buffer condition, the radius of the enclosing sphere is chosen as the largest possible value that does not intersect with any neighboring test boxes, as illustrated in Fig. 4.3. This strategy ensures that the field evaluations at the observation points lie within the far-field zone

relative to the source box.

The optimization problem is then formulated as minimizing the discrepancy between the original RWG-generated field and the field generated by the dipole-based approximation over the surface S [29], and is given by

$$\underset{\beta_{\hat{a},k}}{\operatorname{argmin}} \int_{\bar{r}\in S} \left\| \bar{E}_0(\bar{r}) - \bar{E}_u(\bar{r}) \right\|^2 d\bar{r}, \tag{4.6}$$

where $\bar{E}_0$ and $\bar{E}_u$ denote the electric fields generated by the RWG and uniform dipole-based basis functions, respectively. In practice, the surface integral in (4.6) is approximated by sampling the field at a finite number of N_s points on the sphere, leading to the discretized least-squares formulation:

$$\underset{\beta_{\hat{a},k}}{\operatorname{argmin}} \sum_{l=1}^{N_s} \left\| \bar{E}_0(\bar{r}_l) - \bar{E}_u(\bar{r}_l) \right\|^2 = \underset{\beta_{\hat{a},k}}{\operatorname{argmin}} \left\| \bar{B} - \bar{\bar{A}}\bar{C} \right\|^2, \tag{4.7}$$

where the vector $\bar{B} \in \mathbb{C}^{3N_s \times 1}$ contains the electric field samples generated by the RWG basis functions at the selected observation points. The matrix $\bar{\bar{A}} \in \mathbb{C}^{3N_s \times 3N_d}$ is assembled from block components $\bar{\bar{A}}_{l,k}$, each representing the Green's function-based contribution of the k-th dipole to the l-th observation point. The vector $\bar{C} \in \mathbb{C}^{3N_d \times 1}$ collects the unknown coefficients β_k of the uniform basis functions.

The optimal set of coefficients that minimizes the least-squares error is obtained by solving the normal equations, leading to

$$\bar{C} = \bar{\bar{A}}^{\dagger}\bar{B}, \tag{4.8}$$

where $\bar{\bar{A}}^{\dagger}$ denotes the Moore-Penrose pseudo-inverse of matrix $\bar{\bar{A}}$ [56].

In the proposed surface-based approach, Hertzian dipoles are uniformly distributed over the six faces of each child box. On each face, dipoles are placed on an $n_d \times n_d$ regular grid, leading to a total of $N_d = 6n_d^2$ dipoles per box. Since each dipole carries three vector components corresponding to the x, y, and z directions, the total number of unknown coefficients becomes $3N_d$. This design differs fundamentally from volumetric methods, in which n_d dipoles are placed along each spatial axis, yielding $N_d = n_d^3$.

The implications of this distinction become particularly important in large-scale scattering problems or high-frequency regimes, where the number of boxes increases and finer angular resolution is needed. In both surface- and volume-based methods, each box is populated with a total of $N_d = n_d^3$ dipoles in the volumetric case and $N_d = 6n_d^2$ dipoles in the surface case. When $n_d = 6$, both strategies yield the same dipole count ($N_d = 216$); however, for $n_d < 6$, the surface-based method results in more dipoles, potentially increasing representational fidelity, whereas for $n_d > 6$, the volumetric approach becomes increasingly dense and computationally expensive. Notably, in large problems where each box size a (or equivalently ka) grows, n_d must be increased to maintain the desired accuracy in approximating the Green's function. Following MLFMA-inspired guidelines, N_d is selected using the empirical formula:

$$N_d = \begin{cases} 2(14.14d_0 - 7.17)^2 & \text{if } a < \lambda/8; \quad [57] \\ 2(1.73ka + 2.16(d_0)^{3/4}(ka)^{3/5})^2 & \text{otherwise; } [42] \end{cases} \tag{4.9}$$

where d_0 denotes the target number of accurate digits. Since the volume method scales as n_d^3 and the surface method scales as $6n_d^2$, the surface configuration becomes significantly more efficient in high-n_d scenarios. This efficiency makes it especially attractive for modeling electrically large scatterers, where volumetric methods may lead to excessive memory consumption and computational overhead.

This surface-based configuration requires accurate matching of radiated fields across a surrounding sphere to determine the dipole coefficients. To ensure sufficient angular sampling of the radiated fields, the number of observation points N_s on the matching sphere is set to twice the number of unknown dipole coefficients, i.e., $N_s = 6N_d$. These points are distributed using the spherical Fibonacci lattice, which offers nearly uniform coverage of the sphere. Each point on the sphere, parameterized in spherical coordinates (r, θ, ϕ), lies on a virtual surface of radius $1.5a$, with [29]

$$\theta_l = \arccos\left(1 - \frac{2(l+1)}{N_s}\right), \quad \phi_l = \frac{2\pi l}{\varphi},$$

where $\varphi = 0.5 + 0.5\sqrt{5}$ denotes the golden ratio.

To reduce the runtime computational load, the mapping between subdomain RWG basis functions and uniform dipole basis functions is performed offline during a preprocessing stage. For each subdomain basis function with unit amplitude, the corresponding dipole coefficients are calculated by solving the least-squares problem in (4.8), resulting in precomputed vectors $\bar{C}_n$. During the MVM step of the main simulation, the mapped coefficient vector $\bar{C}$ for a given box is efficiently assembled as a weighted sum of these stored vectors [29] given by

$$\bar{C} = \bar{\bar{A}}^{\dagger} \left(\sum_n \alpha_n \bar{B}_n \right) = \sum_n \alpha_n \bar{\bar{A}}^{\dagger} \bar{B}_n = \sum_n \alpha_n \bar{C}_n, \tag{4.10}$$

where $\bar{B}_n$ contains the far-field electric field samples generated by the n-th RWG basis function with unit excitation. This strategy significantly accelerates the mapping process by replacing expensive projections with simple linear combinations of precomputed components. The details of this first mapping stage is presented in Algorithm 1.

Algorithm 1 Mapping to Uniform Basis Functions [29]

Preprocessing Stage:

Compute $\bar{\bar{A}}^{\dagger}$ in (3.10)

for $i \in$ Non-empty Child Box Indices **do**

for $n \in$ Subdomain Functions in Box i **do**

Compute $\bar{B}_n$ for n^{th} subdomain function.

$\bar{C}'_n = \bar{\bar{A}}^{\dagger}\bar{B}_n$ (via (3.10))

end for

end for

return and store $\bar{C}'_n$'s

During MVM Computation for each Iteration:

for $i \in$ Non-empty Child Box Indices **do**

Initialize $\bar{C}_i = 0$ for the i^{th} box

for $n \in$ Subdomain Functions in Box i **do**

$\bar{C}_i = \bar{C}_i + \alpha_n \bar{C}'_n$ (via (3.12))

end for

end for

return $\bar{C}_i$'s

4.1.2 Translation

Once the mapping to the uniform dipole basis is complete, all non-empty boxes become structurally identical, each containing the same number of dipoles placed at fixed, known positions. This structural uniformity is a critical enabler for efficient evaluation of far-field interactions, as it allows a single set of translation operators to be reused across all box pairs with the same relative displacement vector. Consider two such boxes in the far zone of one another: a source (basis) box centered at $\bar{r}_i$ and a test box centered at $\bar{r}_j$, as illustrated in Fig. 4.4. The relative translation vector between them is denoted by $\bar{w}_{ij} = \bar{r}_j - \bar{r}_i$. Due to

the uniform structure of all boxes, each distinct translation vector corresponds to a unique interaction model that can be precomputed and applied universally to all box pairs with the same relative positioning. This approach reduces computational redundancy and facilitates faster matrix-vector multiplication in the far-field regime. All translation vectors are of the form $a(\hat{x}n_1 + \hat{y}n_2 + \hat{z}n_3)$, or more compactly $[n_1, n_2, n_3]^T$, where each n_k is an integer ranging from -2^L to 2^L.

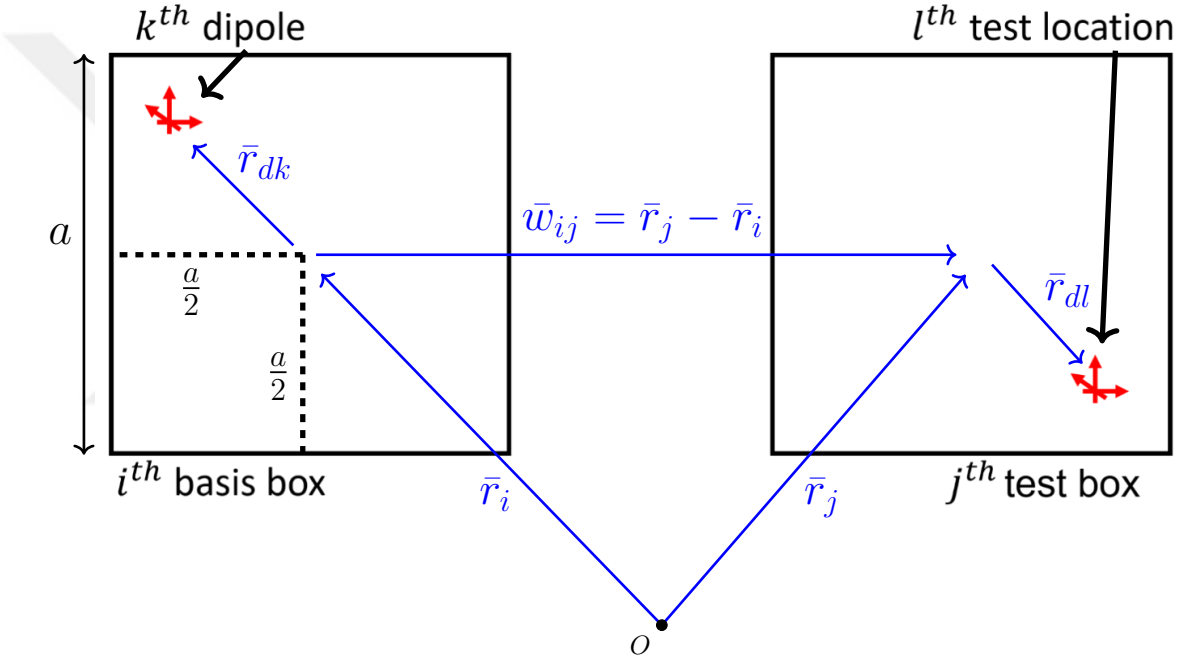


Figure 4.4: Vector definitions used in the translation stage between two boxes in the far-field regime. The k^{th} dipole is located within the i^{th} basis box, and the electric field is observed at the l^{th} test point within the j^{th} test box. Position vectors $\bar{r}_i$ and $\bar{r}_j$ denote the centers of the boxes, while $\bar{w}_{ij}$ represents the translation vector between them. Local dipole positions $\bar{r}_{dk}$ and $\bar{r}_{dl}$ are measured with respect to their corresponding box centers.

Let $\bar{r}_{dk}$ denote the position of the k-th dipole in the basis box relative to its center, and $\bar{r}_{dl}$ the position of the l-th test point relative to the center of the test box. Then, the contribution of the k-th dipole to the electric field at the l-th test point is given by [29]

$$\bar{E}(\bar{r}_{dl}) = \bar{\bar{D}}(\bar{w}_{ij} + \bar{r}_{dl} - \bar{r}_{dk}) \cdot \bar{p}_{\hat{a},k}(\bar{r}'), \tag{4.11}$$

where $\bar{w}_{ij}$ is the relative translation vector from the center of the i-th basis box

to the j-th test box, $\bar{p}_{\hat{a},k}(\bar{r}')$ is the dipole moment of the k-th dipole, and $\bar{\bar{D}}$ is the free-space dyadic Green's function.

To account for all dipoles and test locations between a pair of interacting boxes, we collect the contributions into a matrix-vector multiplication of the form [29]

$$\bar{E}_{i-j}(\bar{r}_{dl}) = \bar{\bar{G}}_{\bar{w}_{ij}}(\bar{w}_{ij} + \bar{r}_{dl}, \bar{r}_{dk}) \cdot \bar{C}_i(\bar{r}_{dk}), \tag{4.12}$$

where $\bar{\bar{G}}_{\bar{w}_{ij}} \in \mathbb{C}^{3N_d \times 3N_d}$ is the translation matrix corresponding to the relative vector $\bar{w}_{ij}$, and $\bar{C}_i$ is the vector of dipole coefficients in the i-th basis box. Each column of the translation matrix corresponds to the contribution of a single dipole to one component of the field at a test point. The total far-zone electric field at the j-th test box is then obtained by summing the contributions from all basis boxes in its far-field interaction list [29] as

$$\bar{E}_j = \sum_{i \in \mathbb{F}_j} \bar{E}_{i-j}, \tag{4.13}$$

where $\mathbb{F}_j$ denotes the set of basis box indices in the far zone of the j-th test box.

Although many translation matrices $\bar{\bar{G}}_{\bar{w}}$ could in principle be derived from one another via rotational and mirror symmetries, the current implementation computes each matrix independently for every relative translation vector $\bar{w}$. This choice simplifies the implementation and avoids the need for additional permutation and sign-matrix logic. However, exploiting these symmetries remains a compelling opportunity for future optimization, as will be discussed in Section 4.3.

Traditionally, the translation matrices are computed using the direct Green's function approach (DGFA), which evaluates the full dyadic interactions between every dipole pair in a source-test box pair. While DGFA ensures high accuracy, it suffers from a computational complexity of $\mathcal{O}(N_d^2)$ per box pair, which can become prohibitive for large-scale simulations. As of now, we do not employ any approximation technique to reduce this cost. However, similar to the strategy adopted in the volumetric method proposed in [29], future work may explore the use of machine learning models to emulate these translation matrices. By learning the aggregate far-field response from physically accurate interaction data, such models could bypass the need for explicit dipole-to-dipole evaluations and

offer significant speedups without compromising accuracy. Algorithm 2, borrowed from [29], presents the details of the translation stage.

Algorithm 2 Translation using DGFA [29]

Initialize $\bar{E}_j = 0$ for $j = 1, \ldots, N_B$ ▷ N_B: # of Boxes
for $i \in$ Non-empty Child Box Indices **do**
  for $j \in$ Non-empty Child Box Indices **do**
    if $\|\bar{r}_j - \bar{r}_i\| = \|\bar{w}_{ij}\| \geq 2a$ **then**
      Calculate $\vec{E}_{i-j}$ defined in (4.12)
      $\bar{E}_j = \bar{E}_j + \bar{E}_{i-j}$
    end if
  end for
end for
return $\vec{E}_j$'s

4.1.3 Inverse Mapping to Subdomain Functions

After the translation step, the total electric field at each test location inside a given box can be computed using (4.13). In order to project this field back onto the original subdomain basis functions, we introduce a reverse mapping strategy that mirrors the first mapping stage, but now works in the opposite direction.

To maintain geometric independence and structural uniformity, we define a set of N_s auxiliary dipoles, referred to here as virtual dipoles, placed on a spherical surface of radius $1.5a$ surrounding each test box. These positions are identical to the sampling points used previously on the Fibonacci lattice, as illustrated in Fig. 4.5. Each virtual dipole is located at a point $\bar{r}_l$ on the sphere, centered around the test box location $\bar{r}_j$.

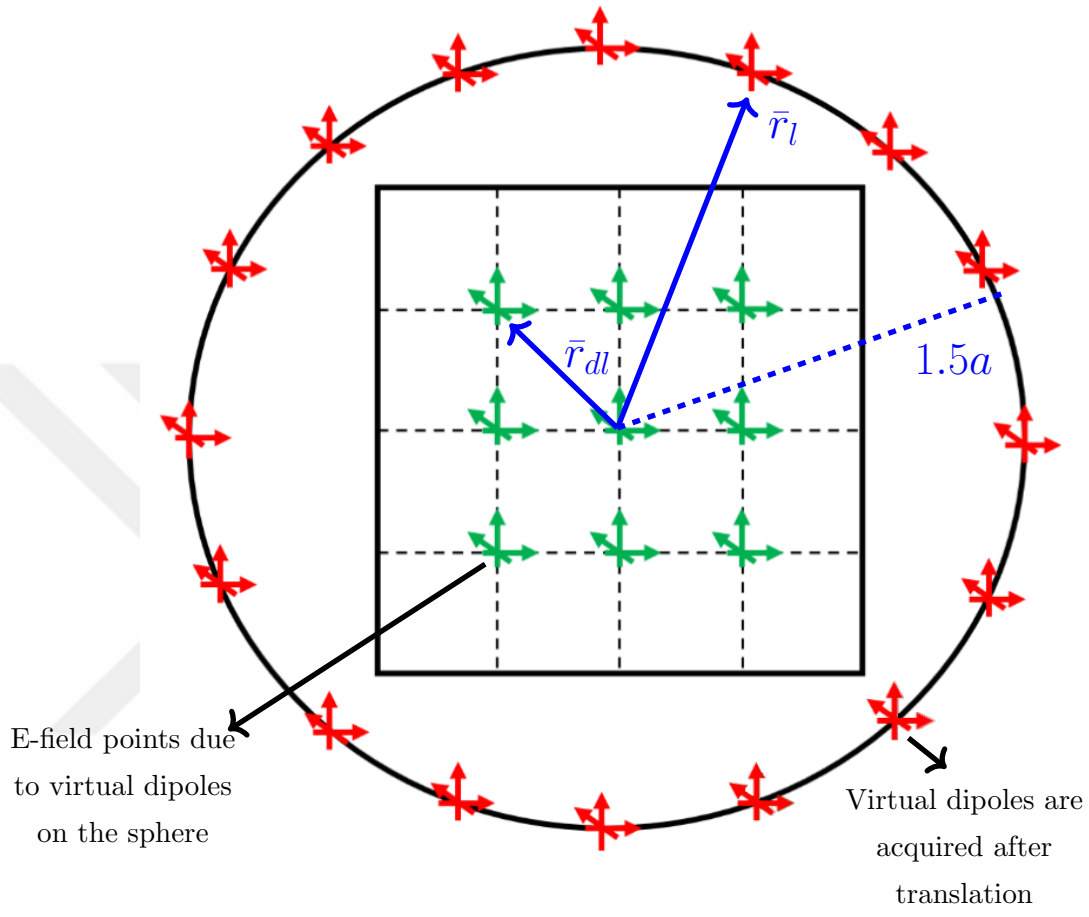


Figure 4.5: Red arrows represent virtual dipoles distributed on the spherical surface, acquired after translation. Green arrows indicate the resulting electric field samples computed inside the test box. This setup illustrates the inverse mapping process, in which far-zone dipole representations are back-projected onto local field samples.

The goal of this stage is to determine the directional excitation strengths $q_{\hat{a},l}$ of the virtual dipoles such that they collectively reproduce the field inside the box, previously computed via translation. This field can be expressed using the following expansion [29]:

$$\bar{E}_{vd}(\bar{r}_{dl}) = \sum_{\hat{a}\in\{\hat{x},\hat{y},\hat{z}\}} \sum_{l=1}^{N_s} \bar{\bar{D}}(\bar{r}_{dl}, \bar{r}_l) \cdot q_{\hat{a},l}\, \bar{p}_{\hat{a},k}, \tag{4.14}$$

where $\bar{p}_{\hat{a},k}$ denotes a unit dipole oriented along direction $\hat{a}$ at the l-th virtual

dipole location, and $\bar{r}_{dl}$ is a point inside the test box. The directional coefficients $q_{\hat{a},l}$ control the contribution of the dipole in the $\hat{a}$ direction at the l-th location.

On the other hand, the target field at these same locations $\bar{r}_{dl}$ is already known from the previous stage and is denoted by $\bar{E}_j(\bar{r}_{dl})$. Therefore, we seek the optimal set of directional coefficients $\{q_{\hat{a},l}\}$ that best match the known field distribution. This leads to a least-squares problem of the form [29].

$$\arg\min_{q_{\hat{a},l}} \sum_{dl=1}^{N_d} \left\| \bar{E}_j(\bar{r}_{dl}) - \bar{E}_{vd}(\bar{r}_{dl}) \right\|^2 = \arg\min_{q_{\hat{a},l}} \left\| \bar{E}_j - \bar{\bar{A}}\bar{Q} \right\|^2, \tag{4.15}$$

where $\bar{Q} \in \mathbb{C}^{3N_s \times 1}$ is the unknown vector of virtual dipole weights and $\bar{\bar{A}}$ is the same system matrix previously defined in (4.7). The least-squares solution for $\bar{Q}$ is obtained as [29]

$$\bar{Q} = (\bar{\bar{A}}^T)^\dagger \bar{E}_j, \tag{4.16}$$

which provides a compact representation of the field inside the box in terms of virtual dipole contributions.

Once the virtual dipole weights are known, the field due to these sources can be evaluated at any arbitrary point inside the test box by inserting $\bar{Q}$ into (4.14). In particular, to evaluate the interaction with a specific subdomain basis function indexed by n, we apply a testing operation over its integration region. Let $\bar{D}'_{jn} \in \mathbb{C}^{3N_s \times 1}$ denote the test vector obtained by evaluating (4.14) at the integration points of the n-th subdomain function. Then, the interaction is given by [29]

$$(\bar{\bar{Z}}_{FF}\bar{\alpha})_n = (\bar{D}'_{jn})^T \bar{Q} = (\bar{D}'_{jn})^T (\bar{\bar{A}}^T)^\dagger \bar{E}_j. \tag{4.17}$$

The vectors $(\bar{D}'_{jn})^T$ are precomputed during preprocessing, in a manner analogous to the precomputation of the $\bar{C}_n$ vectors in the forward mapping stage. The implementation details of this inverse mapping procedure are summarized in the pseudocode provided in Algorithm 3.

Algorithm 3 Inverse Mapping to Subdomain Functions [29]

Preprocessing Stage:

Compute $(\bar{\bar{A}}^T)^\dagger$ in (3.19)

for $j \in$ Non-empty Child Box Indices **do**

for $n \in$ Subdomain Functions in Box j **do**

Compute $\bar{D}'_{jn}$ for n^{th} subdomain function.

$(\bar{D}'_n)^T = \bar{D}^T_{jn}(\bar{\bar{A}}^T)^\dagger$ (via (3.20))

end for

end for

return and store $\bar{D}'_n$'s

During MVM Computation for each Iteration:

Initialize $\bar{\bar{Z}}^{FF}_{\bar{\alpha}} = \bar{\bar{0}}$

for $j \in$ Non-empty Child Box Indices **do**

for $n \in$ Subdomain Functions in Box j **do**

$(\bar{\bar{Z}}^{FF}_{\bar{\alpha}})_n = (\bar{D}'_n)^T \bar{E}_j$ (via (3.20))

end for

end for

return $\bar{\bar{Z}}^{FF}_{\bar{\alpha}}$

4.2 Numeric Results

4.2.1 Volumetric and Surface Methods Comparisons for Mapping to Uniform Basis Functions

In this subsection, the volumetric and surface-based mapping strategies are compared in terms of accuracy and execution time. The goal is to evaluate the trade-offs between these two approaches under varying conditions.

We begin by analyzing the execution time required for mapping subdomain

RWG basis functions to their corresponding uniform dipole representations in both the volumetric and surface-based approaches. This comparison is performed for increasing n_d values ranging from 4 to 12, which directly impact the total number of dipoles used in each box ($N_d = n_d^3$ for the volumetric method and $N_d = 6n_d^2$ for the surface method). The test scenario consists of a single box that contains approximately 37 RWG basis functions. The edge length of this box is set to one wavelength (λ), where $\lambda = 1.2$ meters corresponds to a frequency of 251MHz. The geometric setup for this box configuration is illustrated in Fig. 4.6.

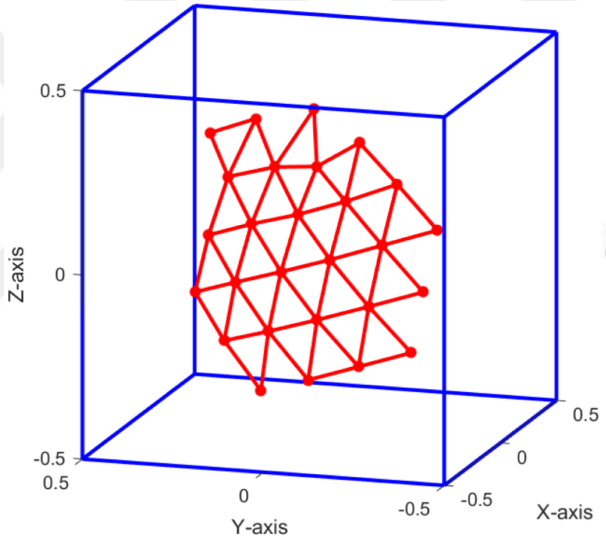
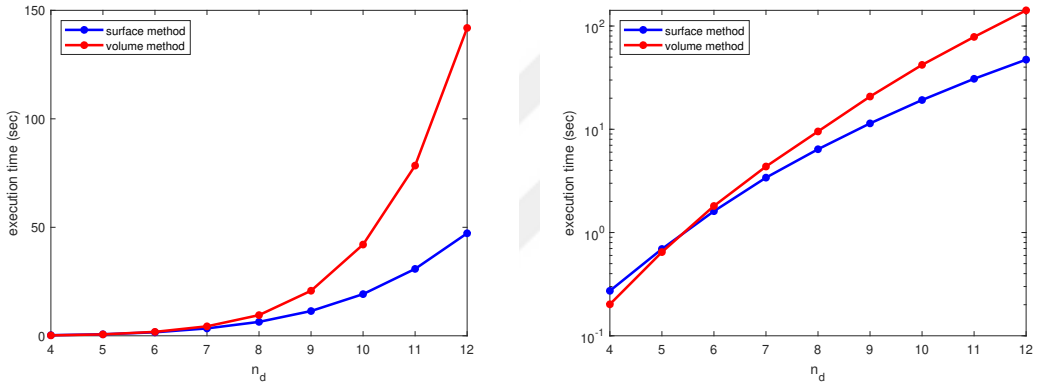


Figure 4.6: Visualization of a densely meshed RWG structure embedded in a unit cube. The triangular mesh is shown in red, with vertices and edges indicating the connectivity of the surface discretization. Although the box appears to span the interval $[-0.5, 0.5]$ along each axis in normalized coordinates, these values correspond to physical dimensions of $[-0.5\lambda, 0.5\lambda]$, resulting in an actual box size of $\lambda \times \lambda \times \lambda$.

For each selected n_d, the least-squares fitting operation described in Section 4.1.1 is executed for all RWG basis functions inside the box, and the total execution time is recorded. These timings include only the preprocessing step required to compute the dipole coefficients, excluding any runtime contributions from matrix-vector multiplications or translation phases. As n_d increases, the surface-based method is expected to demonstrate superior scalability due to its quadratic growth in dipole count, compared to the cubic scaling of the volumetric method as presented in Fig 4.7, where comparison of execution times for both

methods across varying n_d values is provided. As observed in 4.7, the surface-based method consistently outperforms the volumetric approach in terms of execution time, especially as the value of n_d increases. While both methods exhibit growing computational cost with larger dipole counts, the surface method scales more favorably due to its $O(n_d^2)$ complexity, compared to the $O(n_d^3)$ scaling of the volumetric scheme. This difference becomes increasingly pronounced for higher n_d values, highlighting the efficiency of the surface formulation for high-resolution mappings.



(a) (b)

Figure 4.7: Execution time comparison for computing dipole coefficients using surface and volumetric methods for varying n_d. (a) Linear scale, (b) Semilog scale. Box size is $\lambda = 1.2$ m.

A detailed breakdown of execution times for all $(n_d, a/\lambda)$ combinations, including intermediate configurations not visualized in Fig. 4.7, is provided in Appendix Table A.1.

To assess the accuracy of the mapping strategies under controlled conditions, we now consider three distinct configurations where a single RWG basis function is deliberately placed at specific locations inside a unit box as illustrated in Fig. 4.8. These cases allow us to evaluate how the relative position of the source affects the accuracy of the surface-based and volumetric mapping schemes. For each case, the electric field radiated by the RWG function is computed and compared to the field generated by the corresponding dipole-based representation obtained from the least-squares mapping. The comparison is conducted on a far-field observation

sphere surrounding the box, and the error is quantified as the angular-averaged deviation between the two field distributions. All tests are performed at the same frequency of $f = 251\,\text{MHz}$, which corresponds to a free-space wavelength of $\lambda = 1.2\,\text{m}$. The box dimensions are again set proportional to λ, consistent with the previous execution time experiments. The numerical experiments are conducted by varying both the number of dipoles and the box sizes, with the total number of dipoles determined as $N_d = 6n_d^2$ for the surface-based method and $N_d = n_d^3$ for the volume-based method. These configurations are chosen to capture a range of spatial relationships between the RWG source and the dipole distributions. This targeted comparison provides additional insight into the strengths and limitations of each mapping scheme under representative geometric conditions.

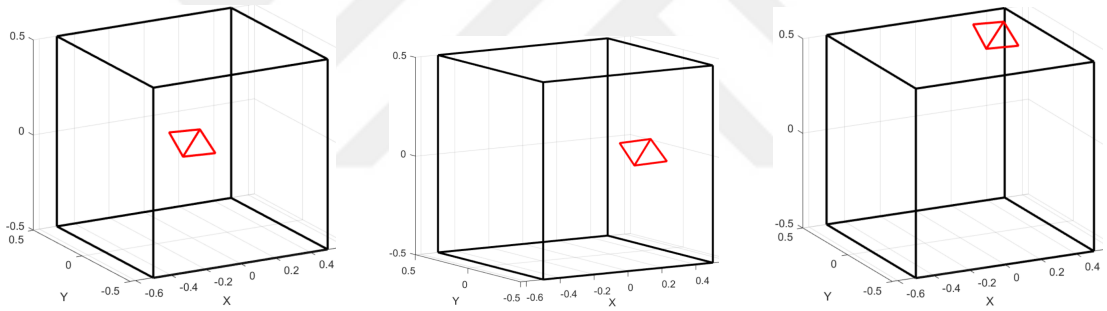


Figure 4.8: Visualization of the RWG basis function location for different test scenarios: (a) The RWG function is located at the geometric center of the box, (b) placed near one of the interior surfaces of the box, and (c) positioned close to one of the interior corners. The box coordinates are shown in normalized units ranging from -0.5 to 0.5 along each axis, corresponding to physical dimensions of $[-0.5\lambda, 0.5\lambda]$ in each direction.

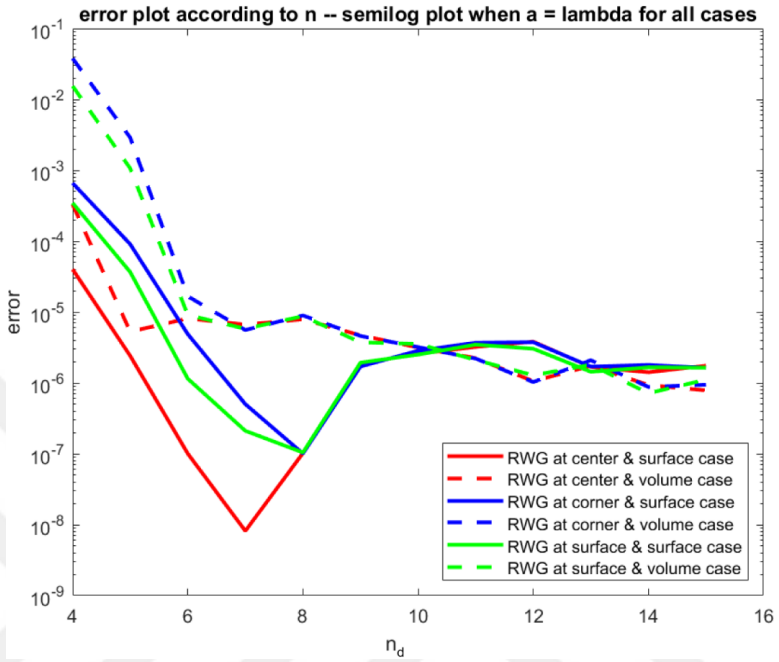
To initiate the accuracy comparison, we first conduct an experiment in which the box size is fixed to $\lambda = 1.2$ meters, while the dipole resolution parameter n_d is incrementally increased from 4 to 9. This setup enables the evaluation of how spatial resolution influences mapping accuracy under uniform geometric conditions. The mapping error is quantified as the normalized residual between the reference electric field vector $\bar{B}$ and the reconstructed field $\bar{\bar{A}}\bar{C}$, computed using the following expression:

$$\text{Error} = \frac{\left\| \bar{B} - \bar{\bar{A}}\bar{C} \right\|}{\left\| \bar{B} \right\|}, \tag{4.18}$$

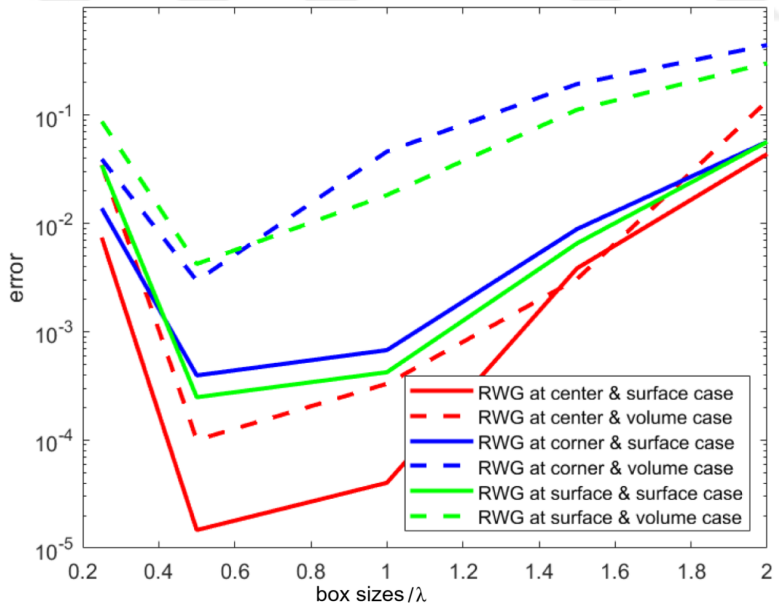
where $\bar{B}$ denotes the reference electric field generated by the RWG basis, $\bar{C}$ is the coefficient vector obtained from the least-squares fitting process, and $\bar{\bar{A}}$ maps dipole sources to far-field observations.

The resulting angular error trends for all three test cases are presented in Fig. 4.9a. The results reveal a consistent advantage for the surface-based method across all configurations. It not only maintains accuracy near the surface but also surpasses the volumetric approach even when the RWG is placed centrally. These findings demonstrate that the proposed surface-based formulation offers robust and superior performance over the volumetric scheme across a range of n_d values and spatial placements.

As a second experiment, we fix the dipole count to $n_d = 4$ and vary the box size from $\lambda/4$ to 2λ. This setup allows us to examine how the accuracy of the volumetric and surface-based mappings responds to increasing spatial domain size under a fixed discretization level. As the box size grows, the distance between the RWG function and nearby dipoles increases, which may affect the fidelity of the mapping. The error trends for all three RWG placements under this configuration are presented in Fig. 4.9b.



(a)



(b)

Figure 4.9: Mapping error comparison for volumetric and surface-based methods. (a) Error across increasing dipole resolutions ($n_d = 4$ to 9). (b) Error across varying box sizes ($\lambda/4$ to 2λ) with fixed $n_d = 4$. Results are shown for three RWG placements.

As seen in Fig. 4.9b, the surface-based method consistently outperforms the volumetric method across all three placement scenarios when the box size increases. This is attributed to the surface dipoles being positioned closer to the RWG in most configurations, especially when the box grows larger. Nonetheless, the overall error levels for both methods remain comparable throughout the tested range, further reinforcing that the proposed surface formulation offers accuracy on par with the volumetric scheme, even under varying geometric scales.

A comprehensive set of numerical results covering all $(n_d, a/\lambda)$ combinations is provided in Appendix Tables A.2, A.3, and A.4, corresponding respectively to the center, surface, and corner placements of the RWG function.

4.2.2 Translation and Inverse Mapping to Subdomain Functions

As discussed earlier, the translated electric field at any observation point within the target box can be computed using the translation and inverse mapping formulations introduced in Equations (4.12) and (4.14), respectively. In this subsection, we assess the accuracy of this process by comparing the translated electric field $\bar{E}_{vd}$, obtained using our dipole-based method, with the original electric field $\bar{E}_{\text{original}}$ directly computed via the Green's function. The relative error between these two fields is quantified using the following expression:

$$\text{Error} = \frac{\left\|\bar{E}_{\text{original}} - \bar{E}_{vd}\right\|}{\left\|\bar{E}_{\text{original}}\right\|} \tag{4.19}$$

To evaluate how this error varies spatially throughout the volume, we compute the electric field and corresponding error at 1000 uniformly distributed test points within the target box. These points are selected to cover the entire cube volume in a structured and regular manner. A visual representation of the sampling grid used for this experiment is shown in Fig. 4.10.

The statistical summary of the computed error values over all sampled points is provided in Table 4.1. As seen in the results, the proposed method achieves high

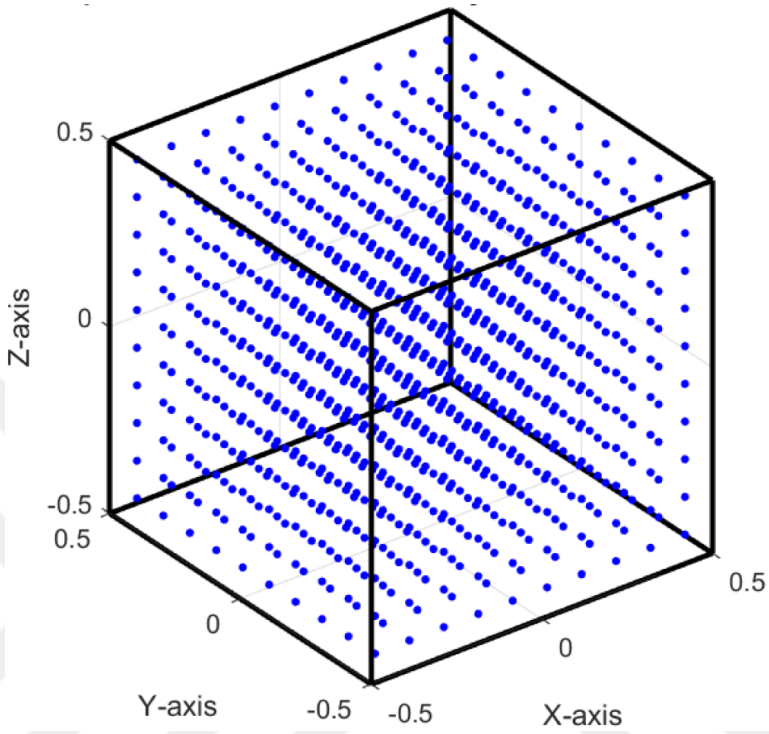


Figure 4.10: Visual representation of the 1000 uniformly placed test points used for evaluating spatial error distribution.

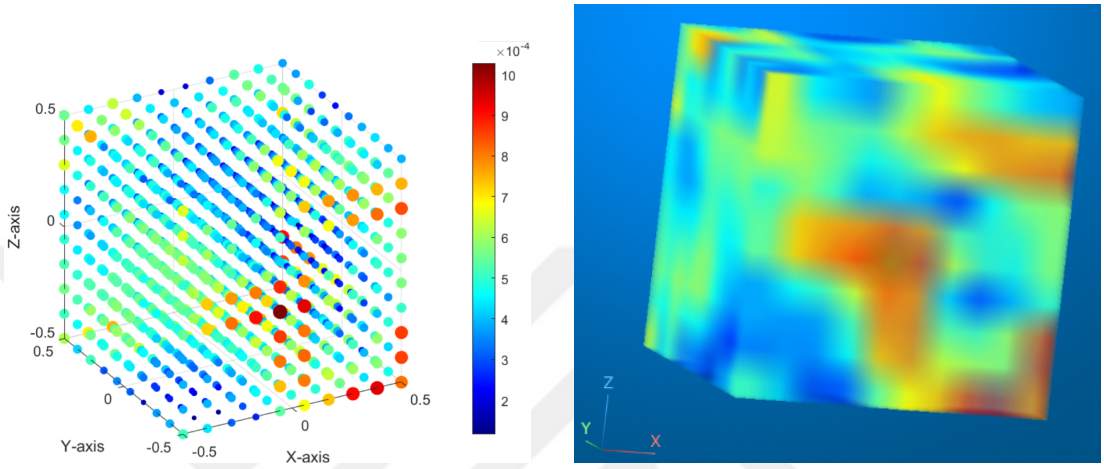
accuracy across the domain, with relatively low average and maximum errors, indicating stable and reliable performance throughout the volume.

Minimum Error	1.1622e-04
Maximum Error	1.0278e-03
Average	4.4886e-04
Standard Deviation	1.3579e-04

Table 4.1: Statistical summary of translation error values computed over 1000 uniformly distributed test points.

The statistical results for these 1000 error evaluations are summarized in Table 4.1, indicating that the proposed method yields accurate results with reasonably low variation. Moreover, the spatial distribution of the error across the box volume is visualized using two different approaches, as shown in Fig. 4.11. The first visualization employs a colormap of discrete test points, while the second provides a volumetric color map rendering of the error field. These visualizations demonstrate that the error remains relatively low throughout the box,

further supporting the accuracy of the proposed translation and inverse mapping methodology.



(a) (b)

Figure 4.11: Colormap-based visualization of translation error across the box volume: (a) colormapped scatter of 1000 uniform points, (b) volumetric rendering of the error field.

These results confirm that the computed translation and inverse-mapped fields exhibit high accuracy throughout the entire box volume. In particular, the maximum error remains below 1.03×10^{-3}, which is considered acceptable for practical electromagnetic simulations, especially given the high fidelity achieved across most regions. This supports the robustness and reliability of the proposed method under typical use cases.

4.2.3 Numerical Results for Full-Scale Scattering Problems

In the previous subsections, the accuracy and efficiency of the proposed surface dipole method were examined on isolated subdomains, focusing on individual boxes and their far-field interactions. These tests validated the fundamental components of the method, mapping, translation, and inverse mapping, under controlled conditions. However, to fully assess the method, it must be applied to

a complete electromagnetic scattering problem where all stages operate together on a full object geometry.

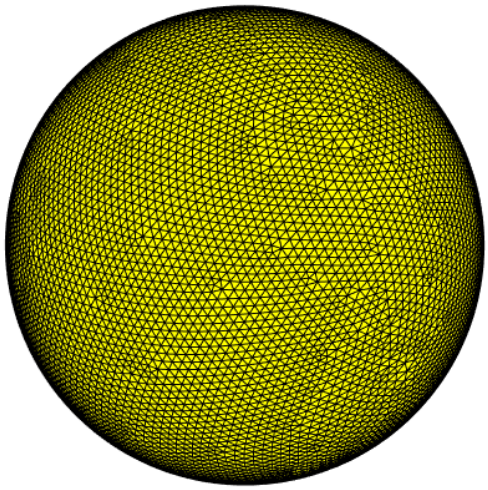


Figure 4.12: Mesh representation of the test sphere using uniform triangulation. The geometry is discretized with RWG basis functions having an average edge length of 12 mm.

The first test object is a canonical perfectly electrically conducting (PEC) sphere of diameter 0.6 m, discretized with 25,995 RWG basis functions. The sphere, shown in Fig. 4.12, is partitioned into multiple boxes, and the complete modeling pipeline is executed to compute the scattered field. The performance of the proposed surface dipole method is compared with three reference solutions, namely the volumetric dipole method, the conventional MoM, and the analytical Mie series. Simulations are carried out for three operating frequencies while keeping the physical mesh fixed, corresponding to electrically small, moderate, and relatively large electrical sizes. This setup allows evaluation of both the accuracy and stability of each method as the electrical size of the object increases.

At 251 MHz, the wavelength is $\lambda = 1.2$ m, making the sphere diameter equal to $\lambda/2$ and the average RWG edge length approximately $\lambda/100$. In this electrically small regime, phase variations across the object are minimal, and the scattered field is relatively smooth. All methods produce nearly identical far-field results, as illustrated in Fig. 4.13.

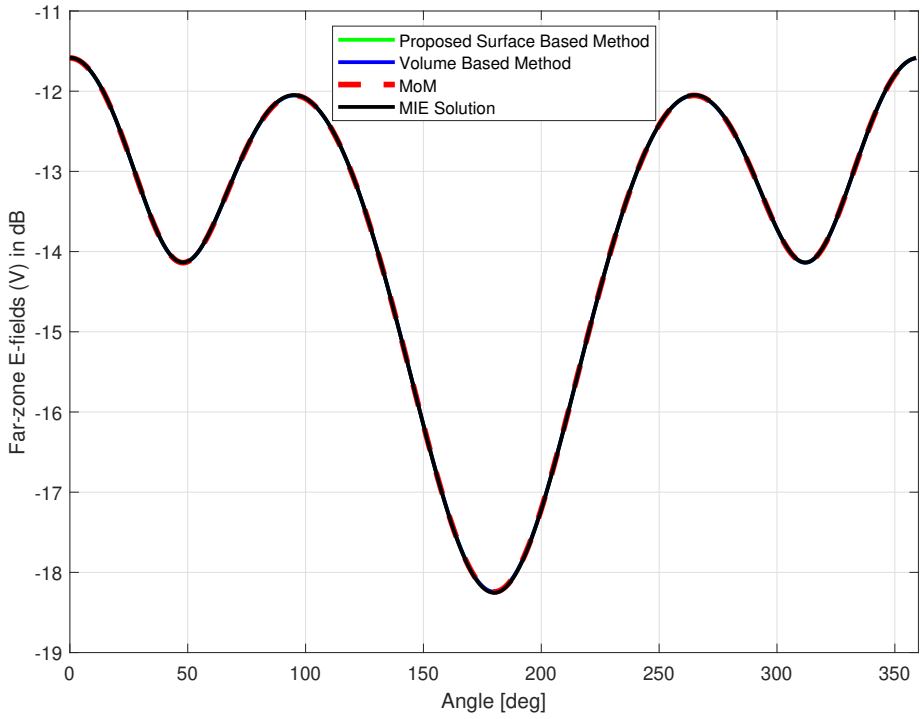


Figure 4.13: Electric field magnitude comparison for a PEC sphere of size $\frac{\lambda}{2}$, discretized using RWG basis functions with an edge length of $\frac{\lambda}{100}$.

When the frequency is doubled to 502 MHz, the wavelength becomes $\lambda = 0.6$ m, so the sphere diameter is now λ and the average RWG edge length is $\lambda/50$. The increased electrical size introduces stronger phase variations, providing a more demanding test of the methods' resolution and stability. Nevertheless, the results in Fig. 4.14 again show excellent agreement among all methods.

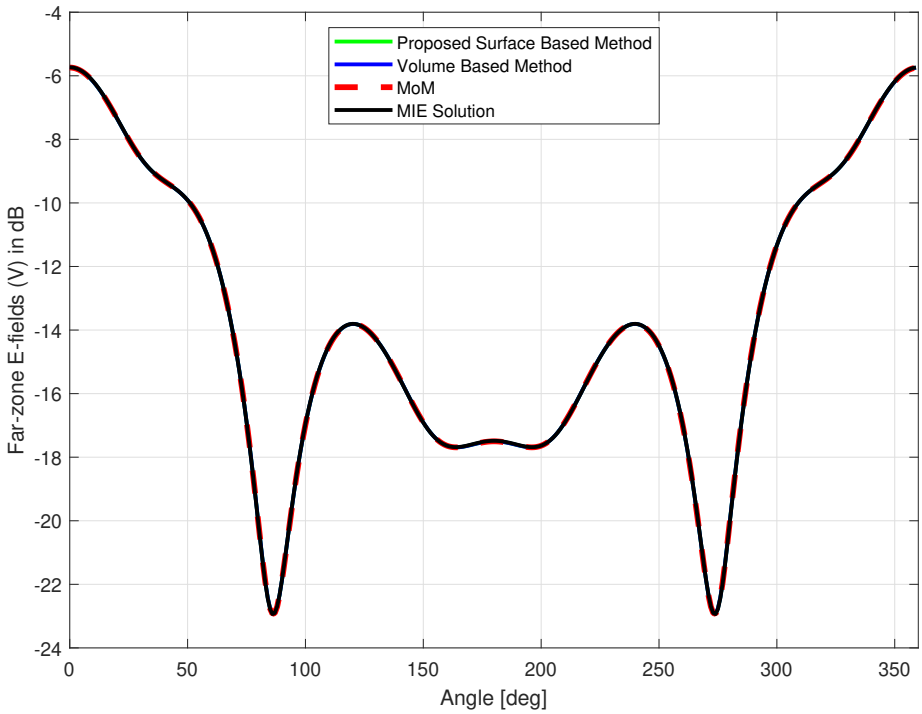


Figure 4.14: Electric field magnitude comparison for a PEC sphere of size λ, discretized using RWG basis functions with an edge length of $\frac{\lambda}{50}$.

Finally, at 1004 MHz, the wavelength is $\lambda = 0.3$ m, making the sphere diameter equal to 2λ and the average RWG edge length $\lambda/25$. In this high-frequency regime, the scattered field exhibits many oscillations and is highly sensitive to numerical accuracy. Despite this, the proposed surface dipole method continues to track the reference solutions almost perfectly, as shown in Fig. 4.15.

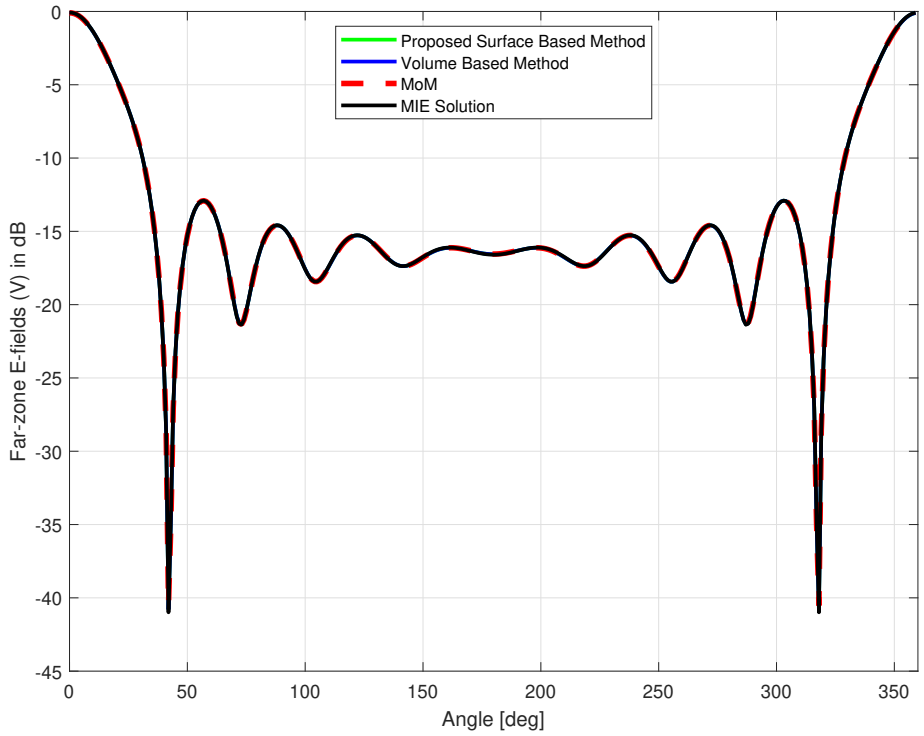


Figure 4.15: Electric field magnitude comparison for a PEC sphere of size 2λ, discretized using RWG basis functions with an edge length of $\frac{\lambda}{25}$.

Across all three electrical-size scenarios, the surface dipole method consistently reproduces the results of the volumetric dipole method, the conventional MoM, and the analytical Mie series with negligible deviation. The agreement holds from the electrically small case, where all methods predict smooth and slowly varying fields, to the electrically large case, where rapid phase variations and fine scattering details are present. Even at the highest frequency, where numerical errors typically accumulate and minor modeling inconsistencies become more pronounced, the surface method maintains virtually identical far-field patterns and magnitudes compared with the reference solutions, confirming its stability, accuracy, and robustness across a wide range of operating conditions.

To further evaluate the robustness of the proposed surface dipole method, we analyze a non-canonical, asymmetric "flamme" PEC object (see Fig. 3.3). Unlike

the symmetric sphere, this case tests the method on irregular shapes where symmetry cannot be exploited and current distributions are more complex. The object measures approximately 0.6 m (2λ at 1004 MHz, $\lambda \approx 0.3$ m) and is discretized into 51,669 RWG basis functions with an average edge length of 3.750 mm ($\lambda/80$). This fine discretization ensures adequate resolution for capturing the oscillatory fields of the electrically large regime. The complete modeling pipeline is applied, and far-zone fields from the surface dipole method are compared with those from the volumetric dipole method and a 4-level MLFMA implementation (Fig.4.16). The far-zone results are nearly identical over the entire angular range, confirming that the approach maintains accuracy on par with established techniques, even for complex asymmetric geometries.

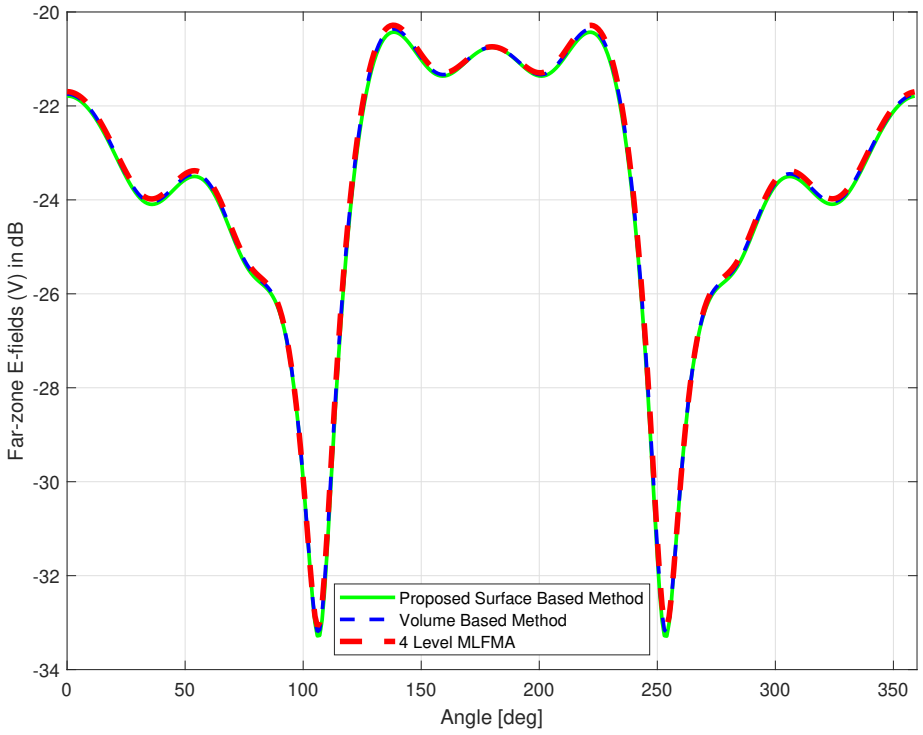
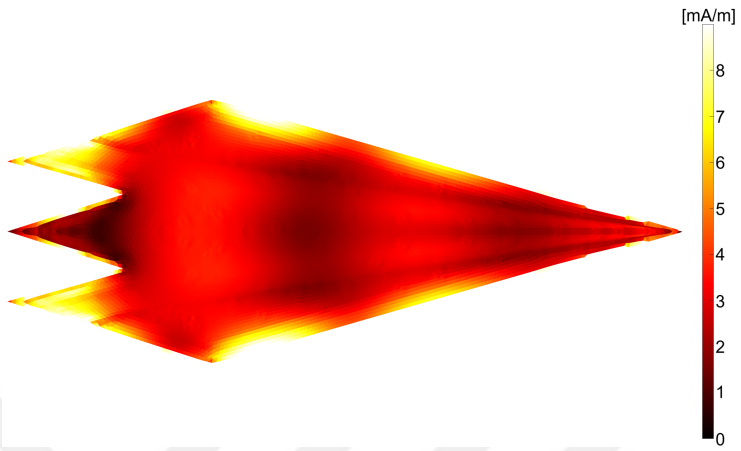


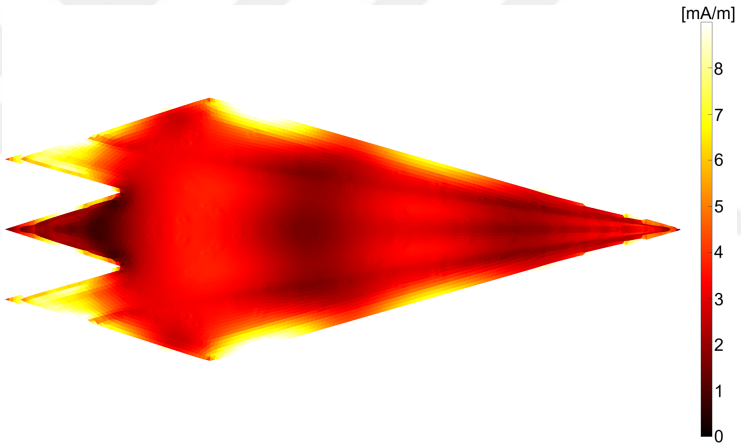
Figure 4.16: Electric field magnitude comparison for a PEC flamme-shaped object of size 2λ, discretized using RWG basis functions with an edge length of $\frac{\lambda}{80}$.

Although far-field agreement is a strong validation metric, it does not fully capture the scattering characteristics, particularly for non-symmetric objects. In such cases, examining the induced current distribution provides deeper insight, revealing localized scattering effects and asymmetries that are not always evident from far-field patterns alone. These distributions highlight regions contributing most strongly to the total scattered field, as illustrated in Fig. 4.17.

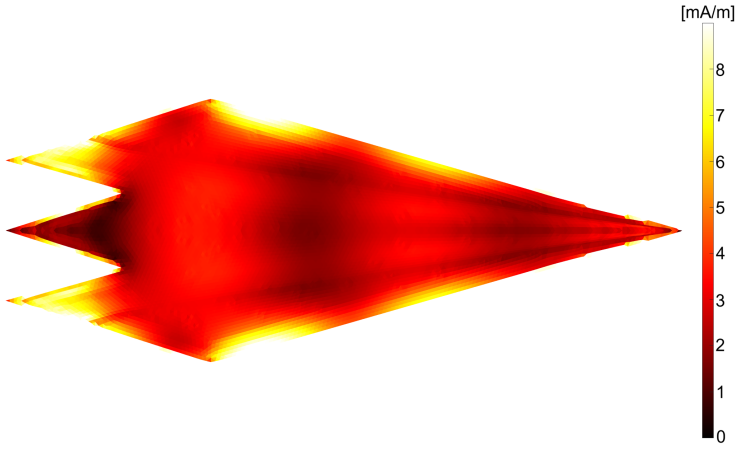
As shown in Fig. 4.17, the surface-based method, volumetric dipole method, and 4-level MLFMA yield virtually identical current-distribution maps. Any differences are negligible and within numerical-noise levels, confirming that the proposed approach maintains the same fidelity as established methods. Such consistency is particularly significant in the electrically large regime, where even minor inaccuracies in translation, mapping, or inverse mapping could otherwise accumulate and degrade accuracy. Across both canonical (sphere) and complex asymmetric (flamme) geometries, the surface-based approach consistently delivers high accuracy and robustness over a broad frequency range. While the volumetric method shows marginal deviations at high frequencies, the proposed surface dipole model remains virtually indistinguishable from well-established full-wave references, confirming its suitability as a highly accurate and efficient alternative to volumetric discretizations for large-scale electromagnetic scattering simulations.



(a) surface method



(b) volume method



(c) 4 level MLFMA

Figure 4.17: Surface current magnitude distributions for the non-symmetric geometry, observed from the top view.

4.3 Future Work for Surface Method

While the current implementation demonstrates the feasibility and accuracy of the proposed surface dipole method, several improvements and extensions remain as promising directions for future research. These include exploiting geometrical symmetries for translation matrix reuse, integrating machine learning for efficient translation modeling, and employing advanced parallelization techniques to further accelerate large-scale simulations.

4.3.1 Exploiting Symmetries in Translation Matrices

In the current implementation, translation matrices $\bar{\bar{G}}_{\bar{w}}$ are independently constructed for each unique relative translation vector $\bar{w}$, without leveraging potential symmetries among box interactions. However, due to the regular structure of the Cartesian grid, many translation scenarios are related to one another through simple geometric operations such as rotations and reflections. These symmetries imply that the corresponding translation matrices are also related via algebraic transformations, involving reordering of indices and sign changes.

By classifying translation vectors into symmetry groups and generating only a minimal set of representative matrices, significant savings in both storage and computation can be achieved. For example, in a level with 316 far-field interactions, only 16 unique translation matrices may be needed if proper symmetry exploitation is applied in volume cases [29]. While similar or even greater reductions may be possible in the surface-based approach depending on the spatial regularity of dipole placement, such symmetry benefits are not inherently guaranteed and can vary depending on the distribution and orientation of the dipoles. Future implementations may incorporate symmetry-aware indexing schemes or lookup tables that map each translation case to its representative matrix and associated transformation.

4.3.2 Machine Learning for Fast Translation Modeling

As discussed in prior work [6, 29], machine learning can be a powerful tool for bypassing the computationally intensive direct evaluation of translation matrices. In the volumetric method, neural networks are trained to emulate the mapping between input and output dipole configurations based on the relative position of two interacting boxes. Once trained, such models can rapidly generate translation matrices without computing individual dyadic Green's function entries.

Although the current implementation does not incorporate machine learning, future extensions of the surface dipole method could adopt similar strategies. By constructing training datasets using high-fidelity dyadic Green's function evaluations, and training lightweight regression networks, one could accelerate the matrix construction process while maintaining acceptable accuracy. This approach is particularly appealing for real-time or many-query scenarios, such as radar cross-section sweeps or optimization problems.

Chapter 5

Conclusion and Future Research Directions

This thesis presented a surface-based fast method for solving large-scale electromagnetic scattering problems, inspired by recent advances in volumetric formulations. The central contribution lies in replacing irregular and geometry-dependent RWG basis functions with uniformly distributed Hertzian dipoles positioned on the surface of the scatterer. This reformulation introduces structural regularity that enables efficient data reuse, reduces implementation complexity, and facilitates compatibility with modern acceleration techniques.

Throughout the study, the surface dipole method was implemented as a standalone fast solver designed to operate without hierarchical structures. Unlike MLFMA, it avoids multilevel tree construction and instead relies on a regular distribution of dipoles over subdomain surfaces. This structural regularity simplifies implementation and enables efficient translation operations using direct evaluations. Comparative simulations demonstrated that the method can produce electric field results consistent with traditional MoM and Mie-series solutions, particularly for canonical geometries like spheres. Furthermore, the use of uniform dipole grids allows for systematic control of accuracy through dipole density and integration refinement.

In addition to the algorithmic formulation, performance optimization was a major focus. The implementation was developed in Python for accessibility and extensibility, and CPU-based JIT compilation was applied to significantly accelerate key routines. Although the resulting performance approached that of optimized MATLAB code in many cases, further improvements are possible.

Looking forward, several research directions can enhance and extend this work. First, the reuse of translation operators via symmetry exploitation remains unexplored in this implementation and could yield substantial memory and speed gains. Second, machine learning techniques may be employed in future to emulate translation matrices or learn system responses more generally, as proposed in recent volumetric methods. Finally, the algorithm is well-positioned for GPU acceleration and large-scale parallelization, both of which are critical for extending applicability to electrically larger and more complex structures.

Throughout this work, `Numba`-based CPU parallelization has been applied to the most computationally demanding routines, especially those involving near-field interactions. Nonetheless, substantial performance gains may be achieved by extending the parallelism to finer granularity and offloading certain tasks to GPUs. Routines such as the evaluation of near-field integrals, translation operator applications, and matrix-vector multiplications in the solver stage are inherently parallelizable. Using frameworks such as `Numba`'s CUDA module, `CuPy`, or `PyCUDA`, these operations can be implemented as custom GPU kernels. Doing so would exploit the massive thread-level parallelism offered by modern GPUs and significantly reduce simulation times. Furthermore, future work could explore hybrid parallelism strategies combining CPU-based multithreading and GPU acceleration. Special attention should be given to optimizing memory access patterns, minimizing host-device communication overhead, and ensuring scalability across hardware configurations.

In summary, this work lays a foundation for a new class of fast surface-based solvers by merging classical electromagnetic theory with modern computational strategies. It bridges the gap between traditional basis-dependent approaches and

uniform representations, opening the door to scalable, modular, and hardware-friendly algorithms for future electromagnetic simulations.

Bibliography

[1] J. M. Jin, *The Finite Element Method in Electromagnetics*. Hoboken, NJ: Wiley, 3 ed., 2014.

[2] R. F. Harrington, *Field Computation by Moment Methods*. Piscataway, NJ: IEEE Press, 1993.

[3] C. A. Balanis, *Advanced Engineering Electromagnetics*. Hoboken, NJ: Wiley, 2 ed., 2012.

[4] C. A. Balanis, *Antenna Theory: Analysis and Design*. Hoboken, NJ: Wiley, 4 ed., 2016.

[5] A. F. Peterson, S. L. Ray, and R. Mittra, *Computational Methods for Electromagnetics*. Piscataway, NJ: IEEE Press, 1998.

[6] S. E. Doğan, "A novel hierarchical machine-learning-based method for efficient solutions of electromagnetic scattering problems," master's thesis, Bilkent University, Ankara, Turkey, 2021. In English.

[7] W. C. Chew, J. M. Jin, E. Michielssen, and J. Song, *Fast and Efficient Algorithms in Computational Electromagnetics*. Boston, MA: Artech House, 2001.

[8] M. Kalfa, "Analysis of slotted sectoral waveguide antenna arrays embedded in cylindrically stratified media," master's thesis, Bilkent University, Ankara, Turkey, 2013. In English.

[9] W. C. Chew, M. S. Tong, and B. Hu, *Integral Equation Methods for Electromagnetic and Elastic Waves*. San Rafael, CA: Morgan & Claypool Publishers, 2009.

[10] M. Takrimi, *Implementation of a Broadband Multilevel Fast Multipole Algorithm for Multiscale Electromagnetics Problems*. Ph.D. thesis, Bilkent University, Ankara, Turkey, 2016. In English.

[11] V. Rokhlin, "Rapid solution of integral equations of classical potential theory," *Journal of Computational Physics*, vol. 60, no. 2, pp. 187–207, 1985.

[12] E. Bleszynski, M. Bleszynski, and T. Jaroszewicz, "AIM: Adaptive integral method for solving large-scale electromagnetic scattering and radiation problems," *Radio Science*, vol. 31, no. 5, pp. 1225–1251, 1996.

[13] J. R. Phillips and J. K. White, "A precorrected-FFT method for electrostatic analysis of complicated 3-D structures," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 10, pp. 1059–1072, 1997.

[14] A. Zhu and S. D. Gedney, "A quadrature-sampled precorrected FFT method for the electromagnetic scattering from inhomogeneous objects," *IEEE Antennas Wireless Propag. Lett.*, vol. 2, no. 1, pp. 50–53, 2003.

[15] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 73, no. 2, pp. 325–348, 1987.

[16] N. Engheta, W. Murphy, V. Rokhlin, and M. Vassiliou, "The fast multipole method (FMM) for electromagnetic scattering problems," *IEEE Transactions on Antennas and Propagation*, vol. 40, no. 6, pp. 634–641, 1992.

[17] Ergül and L. Gürel, "An efficient parallel implementation of the multilevel fast multipole algorithm for rigorous solutions of large-scale scattering problems," pp. 616–619, 2010.

[18] O. Ergul and L. Gurel, *The Multilevel Fast Multipole Algorithm*. Hoboken, NJ: Wiley-IEEE Press, 2016.

[19] M. Kalfa, *Multiple-Precision MLFMA for Efficient and Accurate Solutions of Broadband Electromagnetic Problems*. Ph.D. thesis, Bilkent University, Ankara, Turkey, 2020. In English.

[20] R. Coifman, V. Rokhlin, and S. Wandzura, "The fast multipole method for the wave equation: a pedestrian prescription," *IEEE Antennas and Propagation Magazine*, vol. 35, no. 3, pp. 7–12, 1993.

[21] J. Song, C.-C. Lu, and W. C. Chew, "Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects," *IEEE Transactions on Antennas and Propagation*, vol. 45, no. 10, pp. 1488–1493, 1997.

[22] E. Michielssen and A. Boag, "A multilevel matrix decomposition algorithm for analyzing scattering from large structures," *IEEE Transactions on Antennas and Propagation*, vol. 44, no. 8, pp. 1086–1093, 1996.

[23] E. Darve, "The fast multipole method: Numerical implementation," *Journal of Computational Physics*, vol. 160, no. 1, pp. 195–240, 2000.

[24] Ergül and B. Karaosmanoğlu, "Approximate stable diagonalization of the green's function for low frequencies," *IEEE Antennas and Wireless Propagation Letters*, vol. 13, pp. 1054–1056, 2014.

[25] M. Righero, I. M. Bulai, M. A. Francavilla, F. Vipiana, M. Bercigli, A. Mori, M. Bandinelli, and G. Vecchi, "Hierarchical bases preconditioner to enhance convergence of the CFIE with multiscale meshes," *IEEE Antennas and Wireless Propagation Letters*, vol. 15, pp. 1005–1008, 2016.

[26] B. Karaosmanoğlu and Ergül, "Stabilization of the fast multipole method for low frequencies using multiple-precision arithmetic," in *2014 XXXIth URSI General Assembly and Scientific Symposium (URSI GASS)*, pp. 1–4, 2014.

[27] S. Mittal and J. S. Vetter, "A survey of CPU-GPU heterogeneous computing techniques," *ACM Comput. Surv.*, vol. 47, July 2015.

[28] J. Nickolls and W. J. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, 2010.

[29] E. Koç, M. Kalfa, S. E. Dogan, and V. B. Ertürk, "A novel highly parallelizable machine-learning based method for the fast solution of integral equations for electromagnetic scattering problems." arXiv preprint arXiv:2501.04684, 2025. Available: https://arxiv.org/abs/2501.04684.

[30] R. Lupoiu and J. A. Fan, "Machine learning advances in computational electromagnetics," in *Advances in Electromagnetics Empowered by Artificial Intelligence and Deep Learning*, pp. 225–252, 2023.

[31] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, "Parallel distributed computing using Python," *Advances in Water Resources*, vol. 34, no. 9, pp. 1124–1139, 2011. New Computational Methods and Software Tools.

[32] Z. Fink, S. Liu, J. Choi, M. Diener, and L. V. Kale, "Performance evaluation of Python parallel programming models: Charm4py and mpi4py," in *6th International IEEE/ACM Workshop on Extreme Scale Programming Models and Middleware (ESPM2)*, pp. 38–44, 2021.

[33] E. Bleszynski, M. Bleszynski, and T. Jaroszewicz, "Aim: Adaptive integral method for solving large-scale electromagnetic scattering and radiation problems," *Radio Science*, vol. 31, no. 5, pp. 1225–1251, 1996.

[34] M. A. Yurkin, "Capabilities of the discrete dipole approximation for large particle systems," in *2016 URSI International Symposium on Electromagnetic Theory (EMTS)*, pp. 386–389, 2016.

[35] V. Kindratenko, ed., *Numerical Computations with GPUs*. Springer, Cham, Switzerland, 2014.

[36] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: a LLVM-based Python JIT compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, (New York, NY, USA), Association for Computing Machinery, 2015.

[37] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: A Hands-On Approach*. Morgan Kaufmann, 3 ed., 2016.

[38] W. McKinney, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. Sebastopol, CA: O'Reilly Media, 2 ed., 2017.

[39] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: A structure for efficient numerical computation," *Computing in Science Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[40] M. Cwikla, J. Aronsson, and V. Okhmatovski, "Low-frequency MLFMA on graphics processors," *IEEE Antennas and Wireless Propagation Letters*, vol. 9, no. 9, pp. 8–11, 2010.

[41] S. Rao, D. Wilton, and A. Glisson, "Electromagnetic scattering by surfaces of arbitrary shape," *IEEE Transactions on Antennas and Propagation*, vol. 30, no. 3, pp. 409–418, 1982.

[42] S. Koc, J. M. Song, and W. C. Chew, "Error analysis for the numerical evaluation of the diagonal forms of the scalar spherical addition theorem," *SIAM Journal on Numerical Analysis*, vol. 36, no. 3, pp. 906–921, 1999.

[43] MathWorks, "How MATLAB executes functions — MATLAB & Simulink." Accessed: 2025-07-12.

[44] MathWorks, "MEX files — MATLAB & Simulink." Accessed: 2025-07-12.

[45] W. Stallings, *Operating Systems: Internals and Design Principles*. Boston, MA: Pearson, 8 ed., 2018.

[46] J. Aycock, "A brief history of just-in-time," *ACM Computing Surveys*, vol. 35, no. 2, pp. 97–113, 2003.

[47] I. Corporation, "Intel math kernel library (Intel MKL)." `https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html`, 2023. Accessed: 2025-07-06.

[48] NumPy Developers, "NumPy: The fundamental package for array computing in Python." `https://numpy.org/`, 2024. Accessed: 2025-07-06.

[49] S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997. Available online at dspguide.com.

[50] Y. Liao and D. Roberts, "A high-performance and low-power 32-bit multiply-accumulate unit with single-instruction-multiple-data (SIMD) feature," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 7, pp. 926–931, 2002.

[51] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986.

[52] MathWorks, "GPU computing in matlab." `https://www.mathworks.com/help/parallel-computing/gpu-computing.html`, 2024. Accessed: 2025-07-18.

[53] Numba Developers, "Numba: Accelerate Python functions with JIT." `https://numba.pydata.org/`, 2024. Accessed: 2025-07-06.

[54] PyCUDA Developers, "PyCUDA: CUDA integration for Python." `https://documen.tician.de/pycuda/`, 2024. Accessed: 2025-07-06.

[55] T. Sterling, M. Anderson, and M. Brodowicz, *High Performance Computing: Modern Systems and Practices*. Morgan Kaufmann, 2017.

[56] A. Ben-Israel and T. N. Greville, *Generalized Inverses: Theory and Applications*. Springer, 2nd ed., 2003.

[57] M. Kalfa, V. B. Ertürk, and Ergül, "Error analysis of MLFMA with closed-form expressions," *IEEE Transactions on Antennas and Propagation*, vol. 69, pp. 6618–6623, Oct. 2021.

Appendix A

Full Error Tables for Numerical Results of Mapping to Uniform Basis Functions

Execution Time When Uniform Basis Dipoles on Surface

n_d	$a = \lambda/4$	$a = \lambda/2$	$a = \lambda$	$a = 3\lambda/2$	$a = 2\lambda$
4	0.2729	0.6643	1.5778	3.2423	6.1788
5	0.2718	0.6960	1.6007	3.3741	6.3151
6	0.2725	0.6930	1.6104	3.4009	6.4099
7	0.2708	0.6833	1.5882	3.2965	6.3912
8	0.2728	0.7062	1.6255	3.4053	6.4730
9	10.8457	11.3149	11.4073	11.3248	11.3326
10	18.8889	19.1938	19.2048	19.3107	19.4966
11	30.2835	30.1507	30.8290	30.7634	30.8676
12	47.0984	46.9880	47.2472	47.9061	47.6662

Execution Time When Uniform Basis Dipoles on Volume

n_d	$a = \lambda/4$	$a = \lambda/2$	$a = \lambda$	$a = 3\lambda/2$	$a = 2\lambda$
4	0.2326	0.6373	1.7232	4.1955	9.2913
5	0.2135	0.6537	1.7952	4.3360	9.8982
6	0.2019	0.6466	1.8081	4.3590	9.5296
7	0.2008	0.6457	1.7656	4.2742	9.5090
8	0.2138	0.7123	1.8223	4.3701	9.9332
9	19.8518	20.4091	20.7569	20.3137	20.7408
10	40.6883	41.1461	42.0482	41.6070	41.0801
11	77.6728	78.5806	78.4411	78.0457	77.9423
12	144.6832	145.2092	141.8867	147.5744	144.3971

Table A.1: Execution time values in seconds corresponding to the timing test with a densely populated box containing RWG functions, as illustrated in Fig. 4.6.

Error When Uniform Basis Dipoles on Surface

	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$	$n=9$
$a=\lambda/4$	5.9519e-06	2.4209e-07	3.7156e-08	3.6529e-09	4.8215e-08	9.7217e-07
$a=\lambda/2$	7.8853e-06	3.9973e-07	4.0660e-08	5.7107e-09	9.9018e-08	1.2305e-06
$a=\lambda$	4.0203e-05	2.4146e-06	1.0083e-07	8.1586e-09	1.0217e-07	1.8768e-06
$a=3\lambda/2$	0.0029	4.5375e-05	7.1890e-07	1.7389e-08	6.8789e-08	9.5365e-07
$a=2\lambda$	0.0261	0.0012	1.6006e-05	2.0188e-07	1.9104e-08	1.5968e-06

Error When Uniform Basis Dipoles on Volume

	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$	$n=9$
$a=\lambda/4$	5.4302e-06	1.6110e-07	5.8135e-06	4.1718e-06	1.5025e-06	1.4978e-06
$a=\lambda/2$	1.2948e-05	3.4183e-07	8.8454e-06	2.4562e-06	5.7523e-06	1.6228e-06
$a=\lambda$	3.3023e-04	5.2928e-06	8.1155e-06	6.6515e-06	7.9109e-06	4.6980e-06
$a=3\lambda/2$	0.0054	1.1163e-04	3.2123e-06	4.2724e-06	8.4284e-06	5.3669e-06
$a=2\lambda$	0.1337	0.0016	1.9285e-06	1.2669e-05	1.4132e-05	6.8784e-06

Table A.2: Error values corresponding to the accuracy test where the RWG function is located at the center of the box, as illustrated in Fig. 4.8(a).

Error When Uniform Basis Dipoles on Surface

	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$	$n=9$
$a=\lambda/4$	3.9058e-04	4.0147e-05	4.7623e-06	5.3652e-07	5.5054e-08	7.2438e-07
$a=\lambda/2$	2.4281e-04	3.7818e-05	3.5952e-06	4.3375e-07	7.1626e-08	1.5994e-06
$a=\lambda$	6.6042e-04	9.0811e-05	4.8892e-06	5.0230e-07	1.0088e-07	1.7061e-06
$a=3\lambda/2$	0.0081	4.6035e-04	2.6980e-05	2.2380e-06	1.0287e-06	8.8237e-07
$a=2\lambda$	0.0463	0.0157	4.2838e-04	3.3275e-05	4.7533e-07	1.4997e-06

Error When Uniform Basis Dipoles on Volume

	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$	$n=9$
$a=\lambda/4$	0.0031	2.3488e-04	1.2597e-05	3.0714e-06	1.8827e-06	1.8689e-06
$a=\lambda/2$	0.0035	2.6209e-04	1.0064e-05	3.5018e-06	6.1685e-06	1.7654e-06
$a=\lambda$	0.0380	0.0029	1.6689e-05	5.5778e-06	8.9581e-06	4.6005e-06
$a=3\lambda/2$	0.2013	0.0305	4.1128e-05	3.9537e-06	8.9951e-06	6.8128e-06
$a=2\lambda$	0.4521	0.1317	8.1779e-04	3.2137e-05	1.6536e-05	8.1091e-06

Table A.3: Error values corresponding to the accuracy test where the RWG function is located at the surface of the box, as illustrated in Fig. 4.8(b).

Error When Uniform Basis Dipoles on Surface

	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$	$n=9$
$a=\lambda/4$	1.0647e-04	1.2994e-05	8.8130e-07	1.4363e-07	4.9368e-08	9.8798e-07
$a=\lambda/2$	7.3889e-05	1.3342e-05	7.2390e-07	1.7127e-07	7.5990e-08	1.4572e-06
$a=\lambda$	3.4647e-04	3.6719e-05	1.1477e-06	2.1105e-07	1.0440e-07	1.9355e-06
$a=3\lambda/2$	0.0053	3.0928e-04	8.7290e-06	1.1764e-06	5.9179e-08	1.1127e-06
$a=2\lambda$	0.0392	0.0122	1.4405e-04	2.3062e-05	1.1302e-07	1.0932e-06

Error When Uniform Basis Dipoles on Volume

	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$	$n=9$
$a=\lambda/4$	9.0849e-04	7.2259e-05	3.9582e-06	4.2282e-06	1.9544e-06	2.9007e-06
$a=\lambda/2$	0.0011	8.4947e-05	6.9719e-06	2.6777e-06	7.2435e-06	1.9364e-06
$a=\lambda$	0.0155	0.0011	9.1154e-06	5.8087e-06	8.7462e-06	3.7377e-06
$a=3\lambda/2$	0.0958	0.0120	1.1935e-05	4.4373e-06	8.0433e-06	7.5172e-06
$a=2\lambda$	0.2731	0.0601	2.6746e-04	2.5508e-05	1.6399e-05	6.7609e-06

Table A.4: Error values corresponding to the accuracy test where the RWG function is located at the corner of the box, as illustrated in Fig. 4.8(c).